Modelling Deadlock in Queueing Systems

Geraint Ian Palmer
School of Mathematics
Cardiff University

A thesis submitted for the degree of

Doctor of Philosophy

2018

Abstract

Motivated by the needs of Aneurin Bevan University Health Board, this thesis explores three themes: the phenomenon of deadlock in queueing systems, the development of discrete event simulation software, and applying modelling to the evaluation of the effects of a new healthcare intervention, Stay Well Plans, for older people in Gwent.

When customers in a restricted queueing network become mutually blocked, and all possible movement ceases, that system becomes deadlocked. This thesis novelly investigates deadlock. A graph theoretical method of detecting deadlock in discrete event simulations is given, analytical models of deadlocking systems are built, and these are used to investigate the effect of system parameters on the expected time until reaching deadlock. Furthermore a deadlock resolution procedure is proposed.

An open source discrete event simulation software, Ciw, is developed. This software is designed and developed using best practice principles. Furthermore it permits the use of best practice, such as reproducibility, in simulation modelling.

Ciw is used for the modelling of a healthcare system, in order to evaluate the effect of Stay Well Plans. During the development of these models, a number of techniques are employed to overcome the difficulties of lack of data. Insightful results from these models are obtained, indicating a shift in demand from residential care services to community care services.

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed . . . . (candidate) Date . . . .

STATEMENT 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed . . . . (candidate) Date . . . .

STATEMENT 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed . . . . (candidate) Date . . . .

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed . . . . (candidate) Date . . . .

Acknowledgements

I would like to extend special thanks to my academic supervisors, Prof. Paul Harper and Dr. Vincent Knight, who have mentored me over the past four years. Their discussions have always been interesting, fun, and fruitful, and their guidance has been essential to the completion of this work.

I would like to thank Cathy Brooks for sharing her knowledge and giving advice on understanding the Gwent healthcare system and workforce models. Her support facilitating discussions and meetings with everyone at Aneurin Bevan University Health Board has been invaluable.

In addition to these vital individuals, many more have contributed to this work: I'd like to thank Dr. Julie Vile, Dr. Jennifer Morgan, Dr. Jonathan Gillard, Dr. Will King, who have all given fantastic advice on some technical aspects of this thesis; thanks to Will Beer and Kirsty Jordan for their support and advice from a healthcare and evaluation perspective; and to Dr Tracey England for her support throughout.

Furthermore, without the financial support of the ABUHB this research would not have been possible. I would also like to thank the Phoenix project for offering me the opportunity to share my research and gain some teaching experience at the University of Namibia; and to the Software Sustainability Institute, and the Coleg Cymraeg Cenedlaethol who have supported and funded me in disseminating my research.

I'd like to thank the entire School of Mathematics for their welcoming and supportive environment. Thanks to all at ABUHB who have encouraged, supported, and given advice, guidance and feedback on this work. Thanks to all contributors, users and advocates of the Ciw library, some of whom I have only communicated with online, whose work has been fundamental to this thesis.

Finally thank you to my family and friends, whose patience, help, and welcome distractions have been essential to my completing this work.

Dissemination of Work

Publications

- 2018: **Modelling Deadlock in Open Restricted Queueing Networks**. Palmer GI, Harper PR, Knight VA. *European Journal of Operational Research.* [122]
- 2018: **Ciw: An Open Source Discrete Event Simulation Library**. Palmer GI, Knight VA, Harper PR, Hawa AL. *Journal of Simulation.* [124]

Presentations

- 2015: **Queueing Networks for a Healthcare System Deadlocking, Reinforcement Learning & Workforce Planning**. *EURO 2015, Glasgow.*
- 2015: **Queueing Networks for a Healthcare System, Deadlocking & Reinforcement Learning**. *Young OR 19, Aston.*
- 2016: **Simulating Queues with Ciw**. *PyCon Namibia, Windhoek.*
- 2016: **Using Queueing Networks Modelling to Assess the Impact of the OPICP**. *8th IMA International Conference on Quantitative Modelling in the Management of Health and Social Care, London.*
- 2016: **Llwyrglo yn Rhwydweithiau Ciwio (poster)**. *Cynhadledd Wyddonol y Coleg Cymraeg Cenedlaethol, Aberystwyth.* (Winner best poster).
- 2016: **Deadlock in Queueing Networks**. *CORS 2016, Banff.*
- 2016: **Queueing and Python: pip install ciw**. *PyCon UK, Cardiff.*
- 2017: **Queueing Networks, Deadlock and Healthcare**. *IMA and OR Society Conference on Mathematics of Operational Research.*
- 2018: **Modelling Deadlock in Open Resticted Queueing Networks**. *STEM for Britain, London.*
- 2018: **Ciw: An Open Source Discrete Event Simulation Library (poster)**. *OR60, Lancaster.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes the use of operational research methods in order to model and evaluate the effects of a new healthcare intervention for elderly people, Stay Well Plans, on a healthcare system in Gwent. The healthcare system of interest encompasses secondary care and frailty services offered by Aneurin Bevan University Health Board, and community care services offered by local authorities in the Gwent region of South East Wales.

The journey to fulfilling this objective identified many other interesting research questions and gave rise to some interesting and novel research ventures. As with all research there is an obligation to ensure that the methodology and results are reproducible, and this requirement has driven the software decisions throughout this thesis. This inspired the development of a Python library, Ciw, for undertaking reproducible discrete event simulations. Its purpose is to ensure that best practices are both adhered to in its development and enabled and encouraged in its use; thus rendering reproducible and readable simulation models.

Another intriguing research venture that arose from this work, which is a particular underpinning contribution of this thesis, is the study of the phenomenon of deadlock in open restricted queueing networks. That is the natural and permanent cessation of movement of customers due to mutual, circular blockages. Its applicability to healthcare systems is discussed in Section 1.3. This project proved valuable in the understanding of queueing systems, and thus the modelling task at hand.

This introductory chapter is set out as follows:

- Section 1.1 introduces the Stay Well Plans and the modelling and evaluation objectives.
- Section 1.2 gives an overview of some literature concerning the use of queueing methods in healthcare contexts.
- Section 1.3 establishes the possibilities of deadlock in healthcare systems, and thus the need for a comprehensive study of the phenomenon.
- Section 1.4 formalises the research questions and sets out the structure of the thesis.

1.1 Stay Well Plans

The research described in this thesis is part funded by the Aneurin Bevan University Health Board (ABUHB), and all research has been conducted in close collaboration with healthcare managers in the health board.

The health board [8] covers a geographical area in South East Wales encompassing five counties: Blaenau Gwent, Caerphilly, Monmouthshire, Newport, and Torfaen; as well as some areas of south Powys. It serves over 3 million people, and as of April 2018, the health board employs over 13,000 staff. The health board has a strong association with operational research and the use of modelling to aid and inform healthcare decision, with a team of modellers and analysts embedded within the organisation as part of the Aneurin Bevan Continuous Improvement centre (ABCi) [7].

Traditionally healthcare providers segregate services into isolated pathways: acute or secondary care, primary care, and social or community care. It is becoming accepted now that these are in fact complementary and interdependent, with a push to integrate these pathways. ABUHB are working in partnership with Newport City Council and Age Cymru to deliver an integrated healthcare pathway for elderly people living in Newport. The focus of the pathway is "pro-active patient centred co-ordinated care" [80]; patients at risk of becoming frequent users of healthcare services are offered a single holistic Stay Well Plan (SWP) that intends to use low or no cost services where appropriate.

The main intended outcomes of this integrated care pathway are listed in [80] as:

- Keeping people living safely and independently in their own homes;
- Avoiding unnecessary admission into institutionalised care;
- Developing effective anticipatory care planning, with care wrapped around the individual;
- Developing a continuum of multi-agency provision, which deploys the right resources at the right time in a holistic manner;
- Developing capacity for effective early prevention;
- Developing outcome focused service provision within a community setting as an alternative to primary care.

Prior to commencing this work, a risk stratification tool had been developed by the health board in order to identify those at risk patients to whom the plan would be offered. This tool builds a regression model on 123 statistically significant patient characteristics in order to score the patients' risk factor. These characteristics include age, frequency of occurrences of falls, long term and mental health conditions, and deprivation levels of their residential area. Although a sophisticated tool, there are some concerns over its use. GP practices use the tool to rank their own patients, and are advised to offer the plan to those in the top 3% to 7%; as this ranking happens within the GP practice only there are inconsistencies in the risk level of patients offered the plan across practices. It is known that some practices forgo the tool altogether, offering plans to patients by age, or in a bespoke manner. Furthermore some input variables of the regression model are not independent of one another, with multiple scores for deprivation (WIMD [184] and Townsend Score [112]) and co-morbidity (Charleston Comorbidity Index [46] and QOF Diagnoses [113]), decreasing confidence in its use.

Once identified, patients are offered an integrated care assessment, carried out by a designated Care Facilitator, and, together with the individual and a family member or carer, an anticipatory care plan is developed. This plan may include such interventions as [130]:

- Bathing aid installations
- Nail cutting
- Domestic services
- Fire safety checks
- Hearing aid
- Grab rail installation
- Wheelchairs
- Benefit checks
- Social support
- Befriending

Preliminary data analyses [129, 130] indicates that these interventions may be causing a decrease in demand at a number of healthcare facilities. The aim of this thesis, presented in Chapter 7, is to use simulation modelling and data analysis techniques to determine whether these SWPs are having any significant effect on the demand at the healthcare facilities of interest. In particular predictions about any future effects on demand, due to population growth will be made, and their effects on the workforce requirements at the facilities will be calculated.

1.2 Queueing and OR methods for healthcare

Operational research (OR) is, quoting the website of the UK Operational Research Society, “a way of using analytical methods to help make better decisions” [165].

That is, applying mathematical and computational techniques to decision, managerial, and systems problems. Often particular areas of mathematics are deeply associated with OR, due to their strong aptitude for solving these types of problems. One such area, queueing theory, and the related fields of stochastic modelling and discrete event simulation, are the focus of this thesis.

1.2.1 Queueing systems

The study of queueing systems was established by a paper in 1909 by Agner Krarup Erlang [44]. Here waiting times for call connections in telephone systems were analysed. Queueing theory then remained in the domain of the teletraffic community until the Second World War [83], where mathematicians began solving warfare problems of a stochastic nature, and so developing what we now call operational research. Since then, and especially with the advent of modern computing, the field has become well established with a plethora of useful results and varied application areas [188], including healthcare modelling.

A service centre of a queueing system has a number of elements: the service facility itself containing a number of servers; a queue where customers wait for service; and an arriving stream of customers. A customer arrives according to some arrival discipline A, and waits in line in the queue according to some service discipline Z. Once at the front of the queue, when a server at the service facility becomes free that customer begins service according to some service time discipline B. Once service is complete, the customer leaves the system. This system is described in Kendall's notation [157] by $A/B/c/n/m/Z$, where c denotes the number of servers available at the service facility, n denotes the maximum capacity of the queue, and m the size of the population. The service discipline Z can take a number of forms [157], the most commonly studied being FIFO (first in first out). Commonly studied arrival and service time disciplines include:

- M: Markovian. Customers arrive randomly, and inter-arrival or service times follow an Exponential distribution with parameter Λ, that is they have probability density function

$$f(x; \Lambda) = \Lambda e^{-\Lambda x}$$

For arrivals this is also referred to as Poisson arrivals.

- D: Deterministic. Inter-arrival or service times are non-stochastic and fixed.
- G: General. Customers arrival randomly, and the inter-arrival or service times follow some general probability distribution.

The parameters n, m and Z are optional, taking default values $n = m = \infty$, and $Z = \text{FIFO}$. Common examples include $M/M/1$, $M/D/c$, and $G/M/1$.

Service centres may be arranged in a network, where a customer finishing service at service centre i may have a probability of joining service centre j instead of leaving the system. The following notation for the parameters of queueing systems will be used throughout this thesis; for a service centre i:

- Λ_i denotes the arrival rate of customers to a service centre,
- μ_i denotes the service rate of a service centre (independent of the number of servers),
- $\rho_i = \frac{\Lambda_i}{\mu_i}$ is the traffic intensity of the service centre,
- c_i denotes the number of servers available at the service centre,
- n_i denotes the queueing capacity, that is the maximum number of customers able to queue at any one time, of the service centre,
- r_{ij} denotes the probability of joining service centre j after completing service at service centre i,

and R denotes the matrix of all transition probabilities r_{ij} of a queueing system. In general queueing theory concerns the calculation of some system statistics or performance measures such as the average waiting time, average sojourn time (waiting time plus service time), mean queue length, number of losses, and server utilisation. This can usually involve finding the steady state distribution of the number of customers in the system.

1.2.2 Capturing behaviours

More complicated customer behaviours have been studied, such as baulking, reneging, priory queues, and vacations.

Baulking is the behaviour of an arriving customer deciding not to join the queue, and so getting lost to the system; while reneging is where a waiting customer leaves the queue before entering service. There is a pay-off between two factors that determine why a customer may choose not to join a queue upon arrival: the importance of begin served, and the obstacle relating to waiting in the queue [62]. As it may be impossible for customers to know their future wait, this is usually approximated by the queue length at arrival.

The notion that baulking can cause stability is discussed in [62]. As not all arrivals join the queue, the condition for stability $\frac{\rho}{c} < 1$ no longer applies. In this study, each arriving customer has their own personal queue length threshold for baulking, K, to so that if there are m customers already in the system, and $m > K$ then that arriving customer will baulk. K is modelled as a random variable in order to incorporate individual customer preferences, and the mathematical model concerns $F(m) = \mathbb{P}(K \leq m)$. It is shown that this model, when $K = \infty$ reduces to a queue with no baulking as expected.

Another common method of modelling individual behaviour in baulking is to model this as a probability $b(m)$ of baulking (or equivalently a probability p_m of joining the queue) when there are m customers in the system upon arrival [5, 6, 171, 172]. In most models, $b(0) = 0$, ensuring that no customer fails to join a queue if there is an empty system, that is no customer is entirely queue resistant.

Steady-state distributions for an $M/M/1$ queue where customers exhibit general baulking are derived in [172]. Time epochs of the Markov chain are considered as the time between two service starts. The paper also considers the distribution $Q_m(n)$ the probability that m customers baulk in a time epoch, given than n customers joined in the previous time epoch. In [5, 6] the authors derive steady-state probabilities and mean values for an $M/M/1$ queue under two separate customer behaviours for baulking. In the first baulking discipline customers baulk with probability $\frac{m}{n+1}$ when there are m customers in the system. In the second, customers baulk with probability $\left(1 - \frac{\beta}{m}\right)$ when there are m customers in the system, with $0 \leq \beta \leq 1$ being a measure of willingness to join the queue.

Queues with reneging are usually denoted with an extension to Kendall's notation, by $A/B/c/n/m/Z + D$, where D denotes the reneging method. General reneging is considered in [171]. Here the probability that customer reneges in a time interval Δt when there are m customers in the system is $q_n \Delta t$, provided that Δt is chosen small enough such that no more than one customer reneges in that interval. Another type of reneging is studied in [5] and [6], where the probability that a customer reneges is dependent on the amount of time they have already spent in the queue. In these papers a customer reneges after waiting time t with rate function modelled by $\alpha e^{-\alpha t}$.

Priority queues are those where some classes of customer have higher service priority over others. That is higher priority customers are served before lower priority customers, even if they were entered the queue after them. An overview is given in [177]. In general there are two types of priority disciplines:

- Pre-emptive priorities. Here a customer of a higher priority interrupts the service of a lower priority customer upon arrival. The displaced customer must then wait to continue or restart service.

- Non-pre-emptive, or head-of-the-line priorities. Here lower priority customers are not interrupted; the higher priority arrival is simply placed at the head of the waiting line.

Multi-server priority queues are difficult to analyse, as customers of different priorities may be in service at the same time. Priority queues are classified as one of the service disciplines Z of Kendall's notation [157]. In fact all service disciplines may be considered some form of priority discipline, in FIFO customers arriving before others have priority over those customers.

Queues with vacations are those with interruptions to the flow of services. Vacations are closely related to queueing systems with priority customers, in fact many vacation models can be modelled as priority systems [42], where interruptions are modelled as a second class of customer. A survey of many types of vacations in queueing systems is given in [42]; including:

- Breakdowns: interruptions have pre-emptive priority over regular customers;
- Maintenance: regular customers have pre-emptive or non-pre-emptive priority over interruptions;
- Clock driven schedules: in which servers only return from vacation at specific clock ticks.

More recent developments in modelling vacation systems concern combining these vacation disciplines with other customer behaviours [78]. This includes batch arrivals, multiple vacations, baulking, reneging, bulk services, working vacations (in which servers work during vacations but at a slower rate), and synchronous and asynchronous multi server vacations (in which servers respectively take vacations simultaneously or at separate times).

1.2.3 Networks of queues

The area of queueing networks is the study of a number of service centres arranged in a network, connected by the routing probabilities r_{ij}. A narrative of their development is given in [142], and an overview of the different types of networks studied. Networks can be classified as either **open** or **closed**. Open queueing networks are those that have at least one node to which customers arrive from the exterior, and and least one node from which customers can leave the system entirely. That is the population of customers may be completely replenished. Closed systems are self-contained networks where no customers arrive nor depart to or from the exterior, and as a result always contain a constant total of N customers in the system.

Another classification is that of **restricted** and **unrestricted** networks. In unrestricted networks, there is space for an unlimited amount of customers to wait before a service centre, however in restricted networks there is at least one node with a limited queueing capacity. In these cases customer losses and blocking rules apply. Surveys of the study of queueing networks are given in [41, 94].

One of the earliest papers on networks of queues, [73], concerned the problem of a series of single server Markovian queues in which customers only arrive to the first node, and must be served at every other node in sequence before leaving the system. Although the author uses the term 'phase-type' to describe these services, this system is different from what is now known as phase-type distributions due to the existence of queues and waiting times between each phase of service.

A fundamental result for the study of queues in series is obtained in both [25] and [143], though derived in different ways. The result, known as Burke's Theorem, states that the departure rate of a stable $M/M/c$ queue is equal to its arrival rate. This result is fundamental to the study of Markovian queues arranged in series or networks, as a fraction of one node's departures contributes towards another node's arrivals. The latter paper notes that this result holds for more general service disciplines too, including Last Come First Serve (LCFS), Service In Random Order (SIRO), and systems where the number of servers varies with the number of customers present.

This result in central to the seminal paper by Jackson [71], which studies Markovian, FIFO, open unrestricted networks. It is shown that in steady state, each node, behaves as an independent $M/M/c$ queue, which can be analysed as such, known as the decomposition method. Furthermore system solutions have *product form*, that is for a network of N nodes

$$\mathbb{P}(k_1, k_2, \dots, k_N) = \prod_{i=1}^{N} \mathbb{P}(k_i) \tag{1.1}$$

where k_i represents the number of customers at node i, and $(k_1, k_2, \dots, k_N)$ represents the system state, the configuration of customers around the network. The author extends their original work in [72] where arrival and service rates are dependant on the number of customers in the system.

A significant extension is given in [79], to include in the model different classes of customer, each with their own routing probabilities and service times at each node. In this case each k_i represents a specific ordering of classes of customers in the queue at node i. A few extensions to this model are also presented, including modelling Process Sharing (PS) and LCFS service disciplines, prohibiting service from beginning until there is a certain amount of customers in the queue, and where service effort rely on the current system state.

By setting the external arrival rates to zero, and ensuring rows of the transition matrix R sum to one, this method can be used to model closed queueing networks. A brief offhand remark in [72] gives that certain state dependent arrivals of open queueing networks may also in fact model a specific case of closed systems; and in [86] Jackson's work is adapted to a simple closed system. A focussed study on closed networks, giving similar product form solutions and results, is given in [55], and extended in [17] to encompass different customer classes and a variety of service disciplines (FIFO, PS, infinite server, and LCFS).

Restricted queueing networks are those with no or limited intermediate queues between service stations, where customer losses and blocking rules apply. They are known to be more complicated to analyse than unrestricted systems due to the interdependencies between a node's behaviour and other nodes' states. In these systems, if a customer finishes service at one node but is unable to join a queue at their destination node due to lack of queueing space, that customer remains in the current node, restricting their servers from starting the next customer's service. This is known as blocking.

In [87] two characteristics of the blocking rules are discussed:

1. customers completing service at a blocked station remain there until there is sufficient queueing space at the next station;
2. these customers block other customers from entering that station.

If a service centre has characteristic (1) only then is it referred to as 'classic congestion'. If a service centre has both characteristics then it can be referred to as 'blocking'. In [120] three types of blocking are described:

- Type I blocking occurs when a customer is blocked after completing service, and remains with the server until capacity at their destination node becomes available. This type of blocking is also referred to as 'blocked at service', BAS, or 'transfer blocking'.
- Type II blocking occurs when a customer declares their destination before beginning service, and is only granted service if there is available capacity at their destination node. This type of blocking is also referred to as 'blocked before service', BBS, or 'service blocking'.
- In Type III blocking, instead of becoming blocked, a customer is required to repeat their service if there is no capacity at their destination. This type of blocking comes in two forms, fixed destination where the customer's destination does not change at each repetition of service, and random destination, where the customer's destination is re-sampled from a probability distribution after each repetition. These types of blocking are also referred to as 'rejection

blocking', 'repetitive services', or RS-FD for fixed destinations and RS-RD for random destinations.

For blocking Types I and II above, while the customer is blocked, their server is also blocked, and cannot accept another customer for service. Similarly in Type III blocking a server is busy with a customer until that customer leaves the node. Only Type I blocking is considered in this thesis.

Since blocking introduces interdependencies between nodes, the product form solution of unrestricted networks is not appropriate. A survey of works on these systems is given in [119]. One of the first papers to consider these sorts of systems was [68], which models two nodes in series with infinite, finite and no queueing capacity between service centres. Steady state distributions are derived by defining and solving the systems' difference equations. The same method is used in [13], which investigates two and three node systems, as well as systems with one service centre with infinite queue routing into a number of these two and three node systems.

A two node system with no intermediate queue and blocking is studied in [12]. In this paper moment generating functions of waiting time and number of customers in the system are derived, from which further performance measures can be obtained. A number of blocking situations of a two node system are studied in [91], which are characterised by how the system reaches 'full blocking', and what their 'unblocking rule' is. In this situation the blocking of some servers can effect the ability of other to process customers, such that, given c servers in the first node, if c^* $(1 \leq c^* \leq c)$ servers become blocked all servers cease service, causing 'full blocking'. Then, given that the second node can contain at most M customers, once there are k^* $(0 \leq k^* \leq M + c^* - 2)$ or fewer customers in the system then all services may start again and the first node becomes unblocked. Furthermore this model included a probability of feedback.

Due to the complexities of restricted queueing networks with blocking, exact models may become intractable. Simulation modelling provides an accurate alternative, and various alternative heuristic approaches have been developed to provide approximate solutions. Such a method for solving queueing networks with downstream blocking only is presented in [160]. The algorithm finds the mean values of a queueing network with feed-forward flows and single server nodes. Iteratively working from the node furthest downstream and working backwards, if that station does exhibit blocking it finds an *effective service time*, the weighted sum of service time and the mean time blocked and waiting to transition to the next node, and computes the effective service time for the next upstream node by using a recursive formula. This method is adapted to multi-server queues in [87], and a similar iterative method was used in [88]. Other approximation approaches for restricted networks are given in [4, 36, 121, 132], although arbitrary network topologies pose complications due to the possibilities deadlocks in restricted feedback loops (the heart of this thesis, and discussed here in Section 1.3). The authors of [121] confess that their approximation method has a drawback in this sense, while the approximation method of [132] assume deadlocks all are resolved instantly (though this assumption may not be as trivial as implied: deadlock resolutions are discussed in detail in Chapter 6).

1.2.4 Healthcare applications

Models of healthcare systems vary widely in terms of their contexts; from contained, structured and highly stochastic A&E units, to broad, often not well defined settings such as community, social, and mental health services. It is shown in [51] that there is an inherent requirement to account for variability in healthcare systems in which patients may need to wait, and deterministic models underestimate a number of performance measures. Thus queueing models and simulation lend themselves well to this domain. Their usage for healthcare purposes is widespread, more often used to model services on the more structured and contained end of the spectrum. Overviews of these applications are given in [23, 49, 59, 61, 90, 114].

Some of the first applications of queueing theory to healthcare systems are in [14, 15].

In [14] appointments at an out-patient clinic are investigated using manual (tables of random numbers) Monte Carlo simulation of a queue. The aim here is to consider the trade-off between patient waiting times and wasted times of consultants. Then in [15] a hospital ward is modelled as an $M/M/c$ queue to find the optimal number of beds required; though possible future considerations of priority queues and time dependent arrivals are noted.

In [49] it is seen that a number of behaviours that have been traditionally modelled using queueing theory have been utilised for healthcare problems. These include reneging customers and priority customers at ED, loss models and blocking at hospital wards, and deterministic appointment arrivals. A review of potential queueing applications to pharmacies is given in [114]; and [59] gives health applications of simple queueing models to aid in timely access to facilities and staffing levels. A detailed classification of healthcare applications of queueing models is given in [90], identifying three main uses: design (anticipating the future and allocating resources to meet needs), operations (providing fast and efficient care for the current population), and analysis (gaining information on activities in the current system). Another method of analysing systems of queues is by computer simulation, which allows for the understanding of more complex systems not covered by analytical queueing theory. Surveys of simulation applications to healthcare are given in [23, 61].

Single queue models are utilised in healthcare to model small or isolated systems. A geriatric ward is modelled as an $M/PH/c$ queue in [57], a multi server queue with phase-type service time distribution (indicated by PH in Kendall's notation). Here no queue is allowed to build up, that is patients are sent elsewhere and lost to the system if all c beds are occupied. This model is used to find the optimal number of beds in order to reduce the cost of turning away patients and maintaining empty beds. In [70] specialist clinics are modelled as discrete time queueing systems, in which there are batch arrivals, deterministic bulk services, and probabilistic reneging with retrials.

Queueing networks have been used to model healthcare systems in which patients or jobs flow from one facility to another. The range of system characteristics that have been included in the models emphasise the complexities of healthcare systems, and the often need for bespoke models. A further step is to model facilities across multiple departments, sometimes where there is no communication or coordination. A review of such holistic models is given in [173].

In [3] a health centre is modelled as an open queueing network in order to assess how to improve waiting times. The model contains eight nodes representing five care-givers and three areas prior to seeing a doctor or nurse. The model proved successful and disproved a widely held belief that the front desk was the system bottleneck, concluding that including the time spent prior to seeing a care-giver into appointment times would reduce waits. In [32] the orthopaedics department of the Middelheim hospital in Antwerp was modelled as an open queueing network with five service centres and eighteen patient classes. Both pre-emptive and non-pre-emptive interruptions were incorporated into the service times of one node. Three analytical methods of finding flow times were compared: two formulas applied to the decomposition method, the Kingman and Whitt formulas, and one method used a Brownian model. All models were compared to a simulation model of the system, where the decomposition methods far outperformed the Brownian model.

An open queueing network with multiple classes of customer and non-pre-emptive priority discipline is used in [82] to model an emergency care centre. The priority discipline is applied to some but not all nodes of the network. Here two different priority disciplines are modelled and compared in order to find the most appropriate priority discipline that will ensure waiting time targets are met for each patient class. In [11] an A&E department at a London hospital is modelled as a large scale queueing network, which includes a number of large subsystems. Discrete event simulation is used here to compare a number of different patient priority schemes. An integrated clinical assessment and treatment service in Northern Ireland is modelled in [51] as a network of queues; here discrete event simulation is used in order to capture complexities in staff sharing.

In [87] a queueing network with upstream blocking is used to model the flow of mental health patients through a care system in Philadelphia, where service stations are housing facilities: extended acute hospitals, residential facilities and supported housing. Its aim was to find an alternative approach to the needs assessment approach of capacity planning. An approximation method was used to capture blocking behaviours. Similar iterative approximation methods are used in [24] and [93] to model blocking. Both works model a number of hospital units or wards as restricted queueing networks.

A three node restricted network is used in [37] to model cardiac patients at a medical centre. This model is used to find the optimal number of beds required to both reach target occupancy and reduce refused admissions (customer losses). Simulation is used in [60] to model the restricted network of queues in an Atlanta obstetric unit. A process based approach was used here in order to capture the fact that patients do not repeat services at nodes, despite the presence of feedback loops. This is a detail that may not have been captured in analytical stochastic models.

In [3] the authors list a number of reasons why queueing network models do not match real-life healthcare situations. Queueing network analysis generally focus on steady-state statistics, though in situations where services start and end each day, the system is reset every day and so rarely, if ever, reaches steady-state. Queueing analysis also assumed that inter-arrival and service times are independent of one another, yet if an appointment system is used this is not the case. However, the model used yielded realistic results and some useful conclusions and recommendations. In [87] it was found that the queueing network model with static arrival rates could not effectively model the real-world as customer behaviour exhibited reneging based on waiting times. These may indicate that simulation modelling, which has the ability to capture such complexities, may sometimes be more suitable for healthcare systems.

1.3 Possibilities of deadlock

The development of the Ciw software involved testing its functionality on a number of random queueing systems. This highlighted something what was initially identified as a bug or error in the software, a permanent cessation of activity of customers. In fact there was no error, and instead this testing process had identified a property of some open restricted queueing networks that had not been anticipated: the possibility of deadlock. That is all customers with a server were in a blocked state, waiting on a subset of the other blocked customers to move.

This phenomenon may also arise in models of healthcare systems, where deadlocks may be real or a byproduct of the modelling process. In particular the potential for deadlock may be present in models of the healthcare system of interest in this thesis. A detailed description of the system is given in Chapter 7. Although modelled here with infinite capacities, the real system is a restricted one, with blocking rules applying between facilities. Figure 1.1 illustrates the system to be modelled, showing all possible patient flows between facilities, which contains restricted cycles highlighted in red.

This system contains a self loop (from ED to itself), two cycles of two facilities (between community care facilities and between rapid frailty facilities), and a cycle of three facilities (between the three hospitals). It is pertinent that the facilities within these cycles are related to one another, indicating that communications between the facilities may avert any potential deadlocks. These details may be lost in analytical or simulation models, giving rise to deadlock possibilities

The work of this thesis in Chapters 2, 5 and 6 considers these deadlocking subsystems. One piece of work calculates the expected time to reach deadlock from an empty system. Consider the restricted cycle between the rapid frailty facilities,

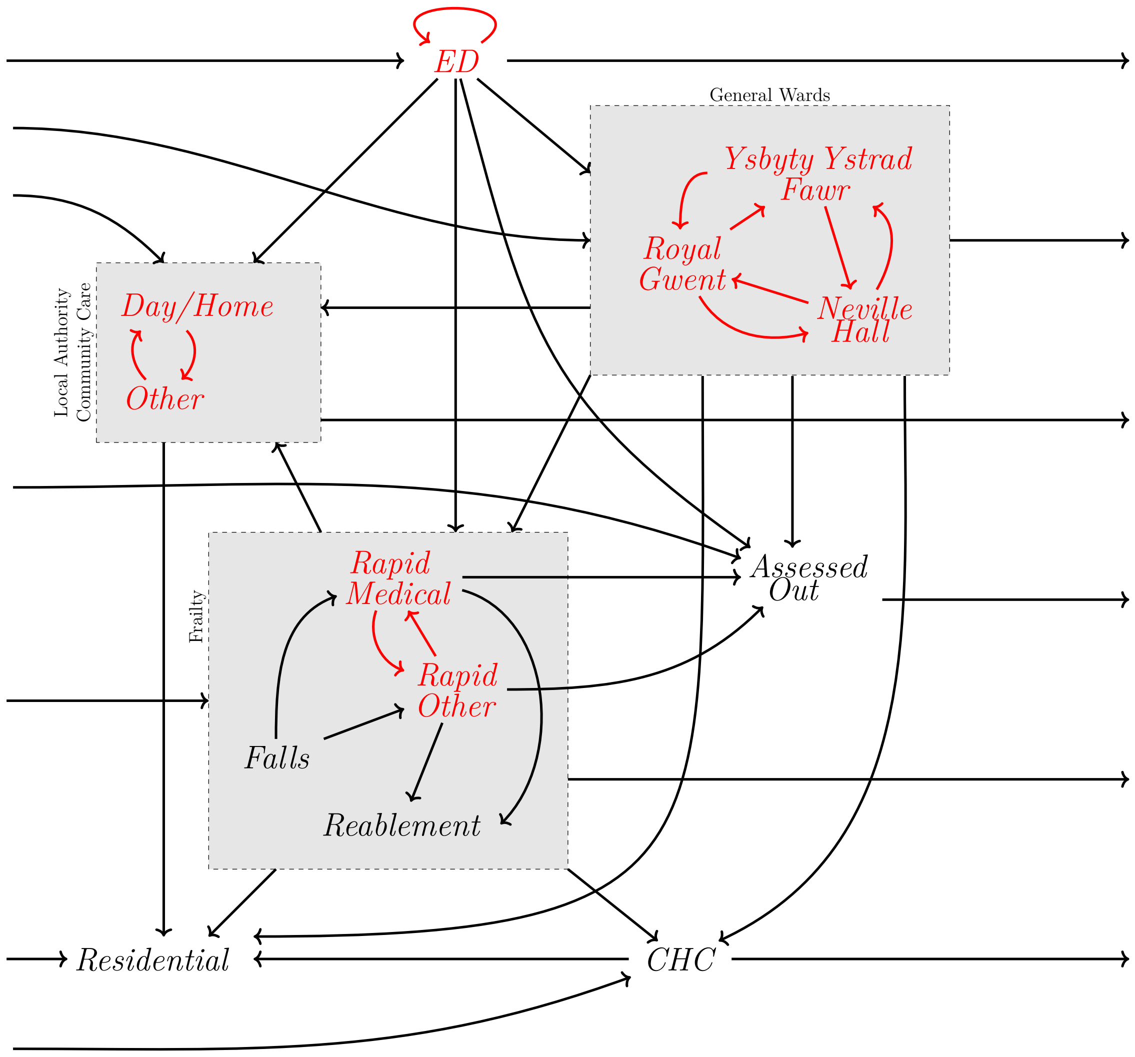


Figure 1.1: Map of the Gwent healthcare system to be modelled, restricted cycles highlighted in red.

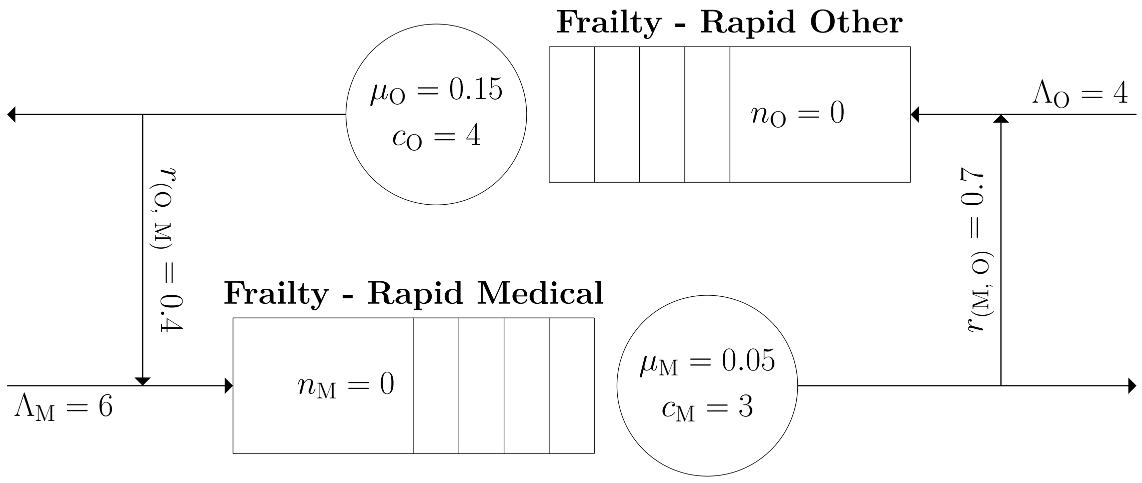


Figure 1.2: A subsystem of frailty services with a restricted cycle.

shown in Figure 1.2. For the purposes of introducing the concept of deadlock illustrative parameters are applied to the system. With these parameters, in time units of days, using techniques developed in this thesis, the system has expected time to deadlock calculated as 185.5 days.

This form of analysis may be useful for systems with real deadlocks. If there is an option to reconfigure the system such that deadlocks are prevented, that is cannot occur in the first place, would this be an worthwhile endeavour? If the expected time to deadlock of the original system is sufficiently large then there may be no need to undergo this reconfiguration. On the other hand, if deadlocks are a symptom of the modelling process only, this must be recognised. Such systems do not have steady states (as deadlocks are absorbing states), causing difficulties in using analytical models and simulation techniques may break down. Therefore an understanding of this phenomenon is imperative for modelling systems with these properties.

1.4 Research questions and thesis structure

This thesis contains eight chapters, together attempting to answer three research questions:

1. What effect have Stay Well Plans had on the healthcare system in Gwent?

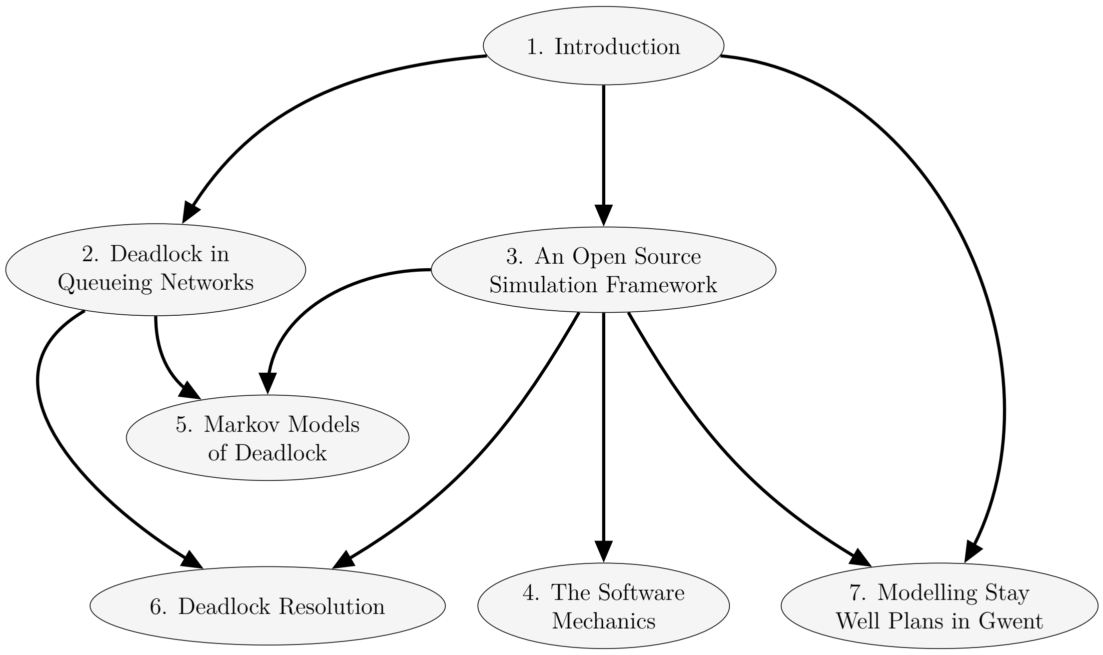

1. Introduction
2. Deadlock in Queueing Networks
3. An Open Source Simulation Framework
5. Markov Models of Deadlock
6. Deadlock Resolution
4. The Software Mechanics
7. Modelling Stay Well Plans in Gwent

Figure 1.3: Structure of this thesis.

2. How can research in the domain of discrete event simulation be conducted in a reproducible manner?

3. What is deadlock and how can it be modelled and investigated?

As stated throughout this chapter, research question 2 stemmed from the need to answer research question 1 in a reproducible manner, increasing confidence in the results and easing its communication. Research question 3 arose due to the testing of the framework developed to answer research question 2, and on closer inspection the presence of feedback loops in the system of interest in research question 1.

The eight chapters of the thesis progress in a logical manner, illustrated in Figure 1.3. An arrow between one chapter and another implies that work from one is used in the methodology of the other. Chapters 2, 5 and 6 strive to work towards research question 3. Chapters 3 and 4 proposes one solution to research question 2, while Chapter 7 presents the work to answer the original research question, question 1. The final chapter, Chapter 8, summarises the work and indicates possible future directions for the work.

This thesis makes use of many mathematical techniques across a number of domains. Methods from combinatorics are utilised in Chapter 2 in order to enumerate configurations of deadlock, and graph theory is utilised in both Chapters 2 and 6 in order to represent and analyse the chain of blockages in restricted queueing systems. Markov modelling in utilised in Chapter 5 and Markov decision processes in Chapter 6 in order to investigate times to deadlock and optimise deadlock resolution procedures. An array of data analysis techniques are employed in Chapter 7 to gain an understanding of the demand at healthcare facilities in Gwent, and to parametrise the simulation models. Discrete event simulation is used throughout to support analytical results, and run three scenarios to evaluate the effect of SWPs.

A summary of each chapter is outlined:

- Chapter 1 has contextualised the three main research questions of this thesis, explaining their origins in terms of the original problem of modelling a healthcare system in Gwent. Background of queueing theory and operational research methods in healthcare settings have been given, and the structure of the remainder of the thesis outlined.
- Chapter 2 introduces the concept of deadlock in open restricted queueing networks. Two types of deadlock are introduced, transient and absolute deadlock, and deadlock configurations are enumerated for complete queueing networks. A graph theoretical method of detecting deadlock in simulation models is given.
- Chapter 3 begins by discussing reproducibility in computational research, and how that applies to discrete event simulation. The chapter chronicles the development of Ciw, an open source discrete event simulation library for Python, designed to enable reproducible simulations.
- Chapter 4 describes the mechanics of Ciw, the simulation approach used, and its implementation. The topic of simultanteous events in discrete event simulation is discussed, and the implementation of the deadlock detection method of Chapter 2 given.

- Chapter 5 presents Markov models for three simple deadlocking queueing systems, which are used to find the expected time to deadlock from an empty system. The systems' behaviours, as some model parameters are varied, is investigated. Finally a bound on the expected time to deadlock of one system is derived, subject to the times to deadlock of embedded queueing systems.

- Chapter 6 discusses the mechanisms of resolving deadlocks in queueing systems. One approach is considered thoroughly, that is creating extra capacity at a particular node and then allowing a constrained sequence of unblockages to occur such that deadlock is resolved. Graph theoretical constructions are given to formalise the language used in these discussion, and results on the particular constraints required to resolve deadlock are given. Finally a Markov decision process is formulated to probe whether finding some optimal resolution policy is a worthwhile consideration.

- Chapter 7 answers research question 1 by presenting a model of a health care system in Gwent, populated by older people. Three scenarios are explored in order to evaluate the effect of Stay Well Plans on this system: SWPs offered nowhere, offered in Newport only, and offered everywhere. Comparing the behaviour of the system across these scenarios allows evaluation of the effects of the plans on the demand and workforce requirements of the modelled facilities.

- Chapter 8 summarises the work of the previous six chapters, and indicates possible directions of future work, identifying further research questions that have arisen.

The work in this thesis has been published in 2 peer reviewed papers:

- [122] which corresponds to work in Chapters 2 and 5,

- [124] which corresponds to work in Chapter 3.

Chapter 2

Deadlock in Queueing Networks

This Chapter introduces the concept of deadlock in open restricted queueing networks. The Chapter is structured as follows:

- Section 2.1 defines and clarifies the concept of deadlock in open restricted queueing networks.
- Section 2.2 reviews how general deadlocks have been discussed in the literature, and how deadlocks in queueing systems have been dealt with when modelling healthcare systems.
- Section 2.3 categorises different types of deadlock that arise in queueing systems.
- Section 2.4 counts the number of deadlock configurations that can arise in complete queueing networks.
- Section 2.5 introduces a general method of detecting deadlock in queueing systems.

Sections 2.1, 2.2 and 2.5 of this chapter have been published in [122].

2.1 Introduction

When a customer becomes blocked, the following definition defines language that can be used to describe their situation:

Definition 1. *If a customer is denied entry to a node due to lack of capacity and forced to remain with their original server, then they are said to be blocked by the customers with servers at that node.*

For the purposes of this thesis, **deadlock**, or a **deadlocked state** of an open restricted queueing network can now be defined as follows:

Definition 2. *When there is a subset of blocked customers who are blocked directly or indirectly by customers in that subset only, then the system is said to be in deadlock.*

This implies that a system is in deadlock when at least one service station permanently ceases to begin of finish any more services, due to circular blocking.

These deadlocks can be real in which case accurate modelling of deadlock is needed, or are unwelcome artefacts of the modelling process that can in fact be adjusted in reality.

Deadlocks arise in many contexts. A simple representation, adapted from [31], of deadlock in the context of road traffic is shown in Figure 2.1. In this situation deadlock, also referred to as gridlock in this context, is caused by the mutual blocking of vehicles. The blue vehicles are blocked from movement due to the position of the yellow vehicles; the yellow vehicles are blocked from movement due to the position of the green vehicles, the green vehicles are blocked from movement due to the position of the red vehicles; and the red vehicles are blocked from movement due to the position of the blue vehicles. Thus the blue vehicles indirectly blocking themselves, and similarly for the other coloured vehicles.

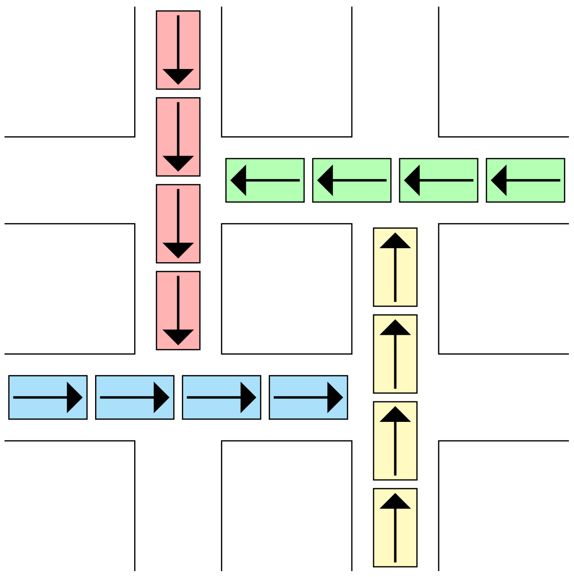


Figure 2.1: A diagrammatic representation of traffic gridlock.

In open queueing networks with at least one cycle containing all service stations with restricted queueing capacity, deadlock can arise. Deadlock is caused by blocking, and this thesis considers Type I blocking only.

Figure 2.2 shows a simple two node queueing network in a deadlocked state. The customer occupying server A_1 has finished service at node A, but remains there as there is not enough queueing space at node B to accept them. The customer at server A_1 is said to be blocked by the customer at server B_1, as they are waiting for that customer to be released. Similarly, the customer occupying server B_1 has finished service at node B, but remains there as there is not enough queueing space at node A, and so the customer at server B_1 is blocked by the customer at server A_1.

When there are multiple servers, individuals become blocked by all customers in service at the destination service centre. Figure 2.3a shows two nodes in deadlock, the customer occupying server A_1 is blocked by customers at both B_1 and B_2, who are both blocked by the customer at A_1. However in 2.3b, customer at A_1 is blocked by customers at both B_1 and B_2, but the customer at B_2 isn't blocked, thus there is no deadlock.

Note that the whole queueing network need not be deadlocked, only a part of it.

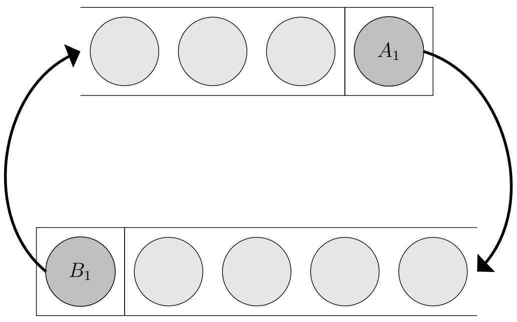
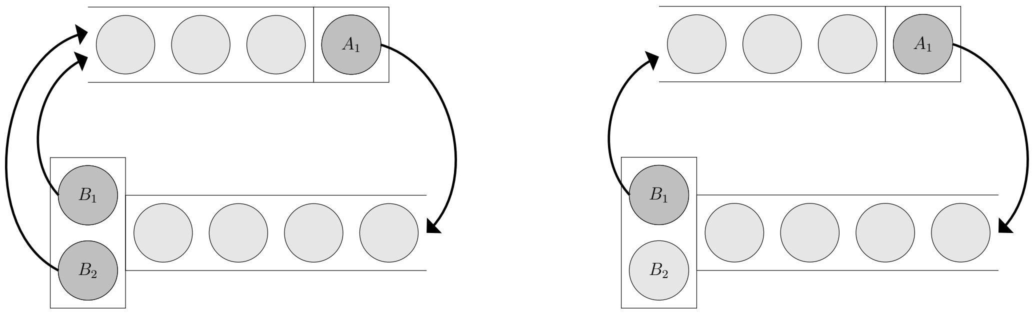


Figure 2.2: Two nodes in deadlock.



(a) Two nodes in deadlock. (b) Two nodes not deadlocked

Figure 2.3: Two nodes: a) in deadlock and b) not in deadlock.

If one section of the network is in deadlock, then the whole system is still said to be in deadlock, even though some customers may still be able to have services and transitions in other areas of the network. This idea is expanded on in Section 2.3.

Consider a further motivating example in the context of a healthcare system: the interface between secondary care services at a hospital and community care services. Patients are admitted to hospital via a variety of routes (for example through emergency services, or outpatients), and via referral from community care services. Patients begin receiving community care packages after referral from GP, or via referral from the hospital. Considering only the hospital and community care services as nodes, this system is shown in Figure 2.4.

If there are no free hospital beds, then patients being referred from community care services will be sustained by community care workers until beds become available. If there are no community care packages available, then patients requiring packages but unfit to return home after a hospital stay will remain in hospital, blocking beds until a community care package becomes available. Type I blocking occurs here, as

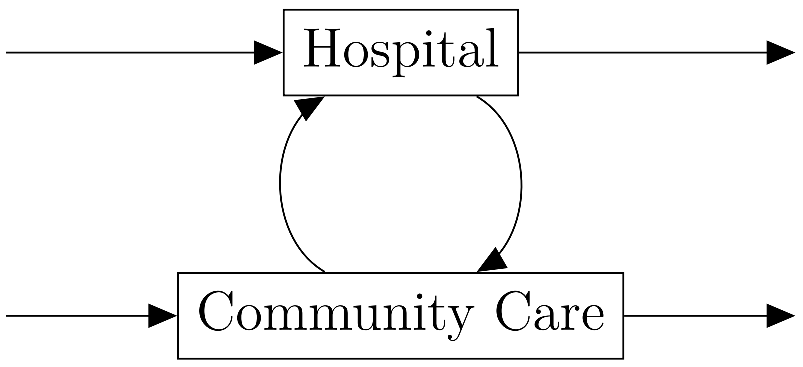


Figure 2.4: Diagram of patient flows at an interface between secondary care services at a hospital and community care services.

patients and staff do not know the future capacity of their next destination prior to service. This type of bed blocking is well known [102]. This causes problems for patients as they are being cared for in an inappropriate setting for their condition, and also for the health care providers as secondary care may be more expensive than community care, and resolution of this causes administrative stress.

In this model there is a non-zero probability of everyone at the hospital blocking beds waiting for community care packages, and everyone at community care being sustained waiting for beds at the hospital. Thus the model will exhibit deadlock. In reality, there is communication between these services and patients can swap places, ensuring no deadlock. So deadlock may be easily resolved in reality, however analytical stochastic models and simulations would be restricted by deadlock. Therefore an understanding of this phenomenon, and an ability to overcome this effect in discrete event simulations, is essential for modelling this system.

2.2 Literature review

Restricted queueing networks that exhibit blocking are well discussed in the literature, both exact [12, 13, 56, 68, 87, 91, 132] and approximate methods [4, 36, 88, 119, 121, 132, 160]. A short description is given in Chapter 1. Discussions on restricted queueing networks with feedback loops, that may exhibit deadlock, are sparse however. In fact the problem of deadlock in queueing networks has either been ignored, not studied, or assumed resolved in much of the literature [119, 121, 132].

Central to the study of deadlock in queueing networks is the concept of blocking. Three types of blocking are described in [120], outlined in detail in Chapter 1.

There has been a body of research around deadlock [31, 144, 145, 154], however no work yet considers the underlying stochastic structure of the system. These types of deadlock are discussed in the context of flexible manufacturing systems and distributed communications systems. Two types are identified: resource deadlocks in which jobs compete for the same resources, and communication deadlocks in which messages are exchanged between limited processes. The systems discussed here are inherently more complex than restricted queueing networks, as resources can be either reusable or consumable, and processes can be using or consuming multiple resources simultaneously, though not necessarily allocated or deallocated to those resources at the same time.

Note that in much of literature discussed here, the term 'deadlock', also referred to as 'deadly embraces' [31], or 'interlock' [111], is used much more broadly than is defined in this thesis. Often, that which is called 'deadlock' is describing circular mutual blockages. That is, the type of blocking shown in Figure 2.3b, where two entities are waiting on one another; however as discussed above this is not considered true deadlock in here, as there is a possibility of this circular mutual blocking resolving itself naturally. These relations may be interesting and problematic in other contexts, however they are uninteresting in the context of restricted queueing networks, as they are standard and regular occurrences. In fact it is often difficult to identify whether circular blocking or deadlock is being discussed in the early literature. Formal definitions are attempted in [176], yet circular blocking is still classified as a deadlock here. The differences between these definitions and those in that paper are discussed in the next section, and summarised in Figure 2.7.

In [31] the following conditions are listed as requirements of a system for deadlocks to potentially occur:

- **Mutual exclusion**: Tasks have exclusive control over resources.
- **Wait for**: Tasks do not release resources while waiting for other resources.
- **No pre-emption**: Resources cannot be removed until they have been used to completion.
- **Circular wait**: A circular chain of tasks exists, where each task requests a resource from another task in the chain.

In open restricted queueing networks the mutual exclusion condition is satisfied as customers cannot share servers; the wait for condition is satisfied due to the rules of Type I blocking; the no pre-emption condition is satisfied in networks that have no or non-pre-emptive priority (this thesis will only considers deadlocks in networks with no priority); and the circular wait condition is satisfied if the queueing network contains a cycle where all nodes have limited queueing capacity.

Allowing a system to reach deadlock can be problematic in cases where automated systems cannot continue operations, or where simulations cannot accurately model reality. In general there are three strategies for dealing with the problem of deadlock [43, 77, 154, 175, 178]:

- **Avoidance**, in which decisions are made as time unfolds to avoid reaching deadlock.
- **Prevention**, in which the system is designed such that in cannot possibly deadlock.
- **Detection and recovery**.

Note that [66] lists the three strategies as prevention, detection and crashing, which is equivalent to having no deadlock strategy. Allowing the system to crash now and again may be more economical in some systems where deadlocks do not occur often enough to justify the investment and effort of implementing an avoidance or resolution strategy.

Prevention and avoidance strategies have been used extensively in an area known as Discrete Event Systems [144, 145]. A distinction between *online* and *offline* resource allocation strategies is made in [175]. In online strategies decisions on resource allocation are made as the system runs, reacting to system's state. That is they are control strategies. Deadlock avoidance strategies are online strategies. On the other hand, in offline strategies all decisions are made at the planning and design stage, before the system runs. Deadlock prevention strategies are offline strategies.

A number of techniques and methods have been used to implement deadlock avoidance, with a survey given in [178]. These techniques generally determine when resources cannot be allocated as that allocation would lead to deadlock. One technique is Banker's algorithm [40, 77] where systems states are categorised as 'safe' or 'unsafe', and jobs are made to wait if allocating resources to them leads to an unsafe state. In [39] a graph is built connecting these system states and the choices that connect the states. This graph helps to avoid making choices that lead to unsafe states. In a similar manner, petri-net models, a type of process mapping [45, 99, 103, 179], and resource allocation graphs [19, 108] have been used to represent permissible and prohibited allocation of resources. In [108] deadlock is avoided by delaying resource allocation if that allocation would cause a cycle in a 'may-wait-for' graph; which represents predicted future resource requests.

In [150] a deadlock avoidance mechanism is proposed for open restricted queueing networks. Here are some external arrivals to some nodes are rejected at some system states, ensuring there is enough capacity for circular mutual blockages not to occur.

The literature has discussed deadlock prevention in closed queueing networks under Type I blocking [89, 100, 119, 150]. Strategies that involve ensuring there will always be enough buffer space or capacity are called conventional prevention strategies in [175]. These strategies for queueing systems have involved determining the minimum queueing space assignment that prevents deadlock for a given population size, or turning customers away if certain nodes are full. A theorem is given in [89] stating that a closed queueing network of K customers, where each node i has c_i servers and queue capacity n_i, is deadlock free if and only if $K < \sum_{j \in C} (n_j + c_j)$ for each cycle C of the network. This ensures that no restricted cycle in the network can fill, thus there is always enough capacity and there cannot be any circular mutual blockages. That paper also presents algorithms for finding queueing capacity allocation that minimises total queueing space and ensures deadlock is prevented, for closed cactus networks. That is networks where no two cycles have more than one node in common. This result is extended to multiple classes of customer in [100], but with restrictions such as single servers and equal service time distributions across customer classes. Here an integer linear programme is formulated to solve this extended deadlock-free queueing capacity allocation problem.

A survey of closed restricted networks is given in [119], where it is noted that extensions to the results in [89] exist for other blocking types. For each type of blocking, the following conditions are required for a closed restricted queueing network to be deadlock-free:

- Under Type II blocking: $K < \sum_{j \in C} (n_j + c_j - 1)$ for each cycle C of the network;
- Under Type III blocking with fixed destinations (RS-FD): $K < \sum_{j \in C} (n_j + c_j)$, similar to the case under Type I blocking;
- Under Type III blocking with random destinations (RS-RD): for a closed network with N nodes, $K < \sum_{i=0}^{N} (n_i + c_i)$, that is the overall queueing capacity cannot exceed K.

It should be noted that open restricted queueing networks are only deadlock-free if they do not contain a cycle.

For simulation modelling however, [176] notes that prevention and avoidance techniques may not be appropriate as they can potentially inhibit realism in the simulation by taking actions that do not occur in the real system being modelled In [48] however, a priority based deadlock avoidance algorithm is implemented in a traffic simulation model. The purpose of this avoidance scheme here is not to reflect deadlock avoidance in reality, but to avoid deadlocks that will occur in the simulation, but not in reality, due to missing information or incomplete models.

A survey of deadlock detection in resource and communication deadlocks is given in [154]. A popular method of detecting general deadlock is the use of wait-for graphs, state-graphs and other variants [29, 30, 31, 38, 43, 66, 108, 174, 175, 176]. These wait-for graphs keep track of all circular wait relations between tasks. Vertices of these graphs may correspond to jobs, resources, or both (bipartite graphs). Edges keep track of waiting, request, or production relations.

One of the earliest instances of constructing wait-for graphs is in [111], for systems where tasks can either request to share or take exclusive control over a resource. Vertices correspond to tasks and edges correspond to the resources being waited for. It is shown that wait-for relationships between resources are transitive order relations. This equally applies to blockages in queueing systems; if customer a is blocked by customer b, and customer b is blocked by customer c, then customer a is waiting for customer c. For the systems described in this paper resources are unique, and deadlock or 'interlock' occurs when a task is waiting on a resource that task is holding (due to the transitive nature of wait-for relations). This corresponds to a cycle in the wait-for graph, indicating deadlock. Similarly in [31] dynamic state-graphs are defined with resources as vertices and requests as edges. Here, although the wait-for graphs may also apply to systems with more than one of each resource type, only in systems where resources are unique can deadlock be found. That is when the state-graph contains a cycle.

In [105] 'transaction-wait-for' graphs (TWF) are named, in which vertices correspond to transactions, and edges correspond to a situation where one transaction is waiting on another transaction to complete before being able to begin. The presence of cycles in the TWF are sufficient to highlight deadlock, as multiple out edges denote waiting on all successors to complete their task. This method is however known to be unreliable [43]. In [30] 'simple bounded circuits' are defined by giving the vertices and edges of the state graph labels in relation to a reference node. The existence of these circuits within the state graph indicates if the system is in deadlock. A survey of a number of deadlock detection algorithms is given in [43], all of which use a form of wait-for graph, some of which store numerous bits of information by colouring or labelling edges or vertices. Most algorithms here detect cycles in the wait-for graphs, but many are reported to be unreliable, failing to detect some deadlocks and detecting some phantom, non-existent deadlocks. Most of the algorithms discussed in that survey do not detect deadlock as defined in this thesis, in Definition 2, but simply find cases of circular blocking. These are still useful however, as circular blocking is a prerequisite for true deadlock, and may still be interesting or problematic in their original contexts.

Bipartite entity-resource graphs (E/R graphs), sometimes referred to as wafer-chamber (WC) graphs, a type of wait-for graph, are used in [38, 66, 174, 175] to detect deadlock in systems with both consumable and reusable resources. The vertices of these E/R graphs correspond to processes, types of reusable resources, and types of consumable resources; while edges correspond to resource requests, resource assignments, and the capabilities of processes to produce consumable resources. It is shown in [66] that if processes can only request one resource, then the system is in deadlock if the E/R graph contains a knot. Two different types of deadlock are defined in [38], transient deadlock and permanent deadlock (note that these are not the same types of deadlock discussed in this chapter); detected by finding strongly connected components and knots in the E/R graph respectively. A deadlock resolution procedure is proposed that attempts to break cycles in the entity-resource graph. This work is furthered in [176] where deadlock is detected and resolved for situations where entities may request more than one resource.

Deadlock detection and recovery is listed as one of the two possible solutions for handling deadlock in queueing networks in [2], although there is no further discussion on the matter. Deadlock detection and recovery in closed queueing networks by swapping customers is assumed in [132], with zero transition time assumed between deadlocked states and the corresponding resolved state. Time to resolve deadlock may not be negligible in reality. The assumption that deadlocks are detected and resolved instantaneously in models of queueing networks is fairly common [87, 88, 121, 132]. However, not only are instantaneous deadlock resolutions unrealistic, it is shown in Chapter 6 this resolution isn't as straightforward as these papers imply.

Queueing systems with restricted feedback loops exhibiting mutual blocking, such as the one described in the previous section (Figure 2.4) have been observed in real healthcare systems. The strategies used to overcome, or ignore, deadlock in these examples emphasise the discrepancies that occur between common modelling techniques and reality in systems that may reach deadlock. Two major techniques have been identified: adjusting the model so that the deadlock phenomenon is prevented, and to overlook deadlock altogether.

In [121] a model of patients flows around Geneva University Hospital is built, where restricted feedback loops exist, and so the possibility of deadlock. However the authors here state that this mutual blocking of this kind "may be irrelevant in practice given that the swapping of patients can be identified and carried out easily", so an approximation method can be used to account for the blockages. In [87] a health and community care system is described as having restricted feedback loops. However due to ease of modelling, and to avoid the restrictions caused by deadlock, these feedback loops are omitted from the model. The same tactic is used in [37] to model cardiac patient flows, where one vital patient routing is omitted from the model to reduce complexity. In some cases blocking is ignored completely despite there being restricted feedback loops in the real system, such as the model of a London A&E unit in [11]. These examples highlight the scale of the problem with modelling restricted feedback loops, with some modellers compromising model accuracy and realism in favour of model simplicity and avoiding the problem.

More worryingly some models, such as that of an Atlanta obstetric unit in [60], ignore the possibility of deadlock altogether, despite acknowledging both blocking and the existence of feedback loops in the model itself. Here simulation is used. Clearly the model did not run into any difficulties, however a lack of consideration for the phenomenon may indicate that an inaccurate conceptual model way used. A hospital in the United States is modelled in [24], encompassing intensive care, step-down, acute care, and post-acute care units, with cycles in the routing of patients. An iterative decomposition approximation method is developed to model the system as an open restricted queueing network with blocking, incorporating expected blockage times into a effective service rate. However any deadlock possibilities are ignored. This indicates that the approximation method used may be inaccurate. A similar iterative approximation is used in [93] to capture the bed-blocking relationships between hospital wards. Again, despite the presence of a restricted cycle, the concept of deadlock is overlooked, raising doubts to the accuracy of the approximation method.

2.3 Types of deadlock

The term deadlocked is used here to describe a queueing network in deadlock, a node that is part of the deadlock, and components of the network where all nodes are experiencing the same deadlock. For the purposes of this work, let's define a queueing network as **connected** or **complete** according to the following definitions.

Definition 3. *A queueing network is connected if there is a possible path from every node to every other node regardless of direction.*

Definition 4. *A queueing network is complete if, from any node, there is a non-zero* *probability of joining any other node of the network directly, including re-entering the same node.*

Figure 2.5 illustrates these concepts. Note that a connected network is equivalent to the underlying directed network being one connected component, and a complete queueing network is equivalent to the underlying network being a complete digraph with loops. Thus a complete queueing network is connected.

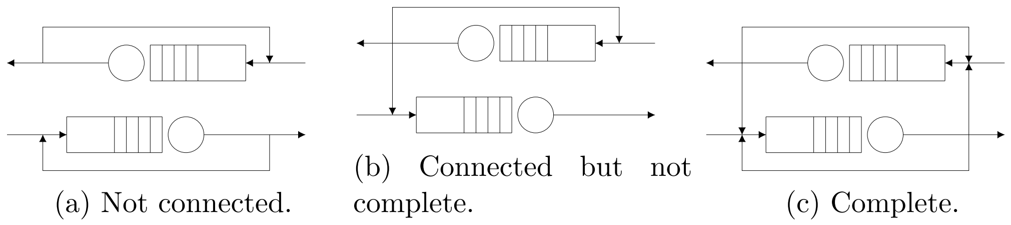

(a) Not connected. (b) Connected but not complete. (c) Complete.

Figure 2.5: Illustrating connectedness and completeness in queueing networks.

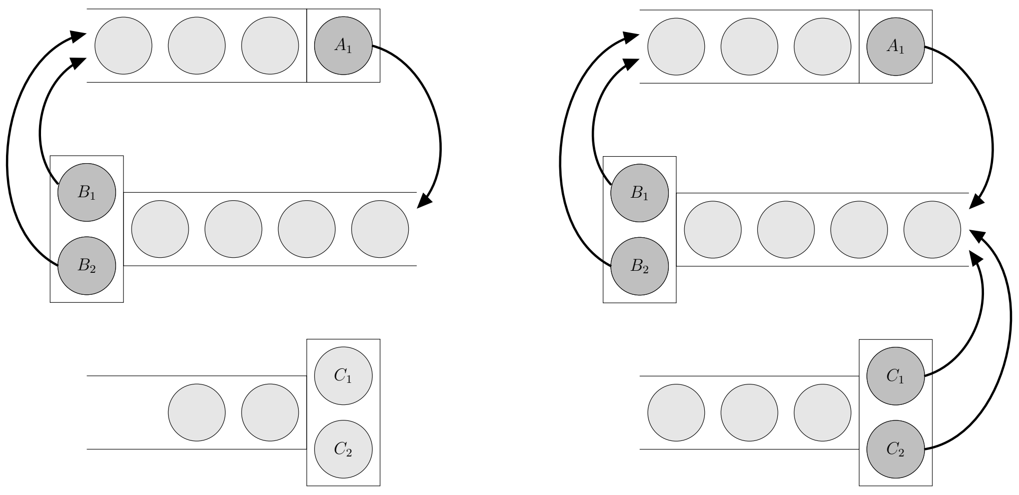
The different configurations of which nodes experience deadlock can be thought of as different types of deadlock. For connected queueing networks, these deadlocks can be classified into **transient deadlock** states and **absolute deadlock** states, defined below:

Definition 5. *A transient deadlock state is when there are still some potential changes of state whilst a subgraph of the queueing network is itself in deadlock.*

Definition 6. *The absolute deadlock state is when all subgraphs of the queueing network are in deadlock.*

If a system is in transient deadlock, there are still some customers who have the potential for movement, however there are a subset of customers who are permanently blocked. If a system is in absolute deadlock, then all customers are permanently blocked.

Figure 2.6 shows a three node network in both a transient and absolute deadlock state. In Figure 2.6a the occupants of servers B_1 and B_2 are blocked from entering node A; and the occupant of server A_1 is blocked from entering node B, and so these two nodes are in deadlock. However, node C can continue with regular services, until the occupants of every server of C attempt to join a deadlocked node. At which point, the whole system is deadlocked, and so has reached absolute deadlock, shown in Figure 2.6b.



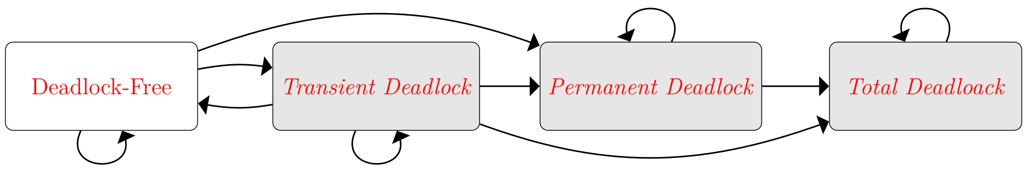
(a) Transient deadlock.

(b) Absolute deadlock.

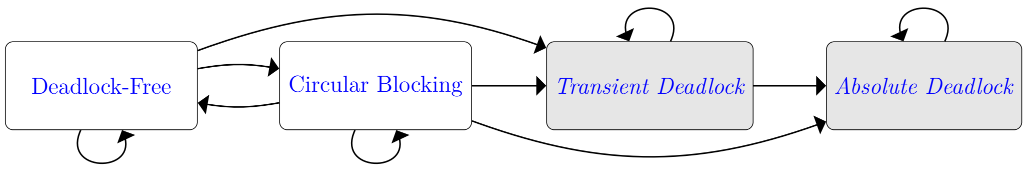
Figure 2.6: Types of deadlock: a) transient and b) absolute.

If the queueing network is complete, then there is a definite probability that once one part of the network is in deadlock, all nodes will fall into a deadlocked state, simply by the individuals in the non-deadlocked nodes attempting to transition into a deadlocked node, or those non-deadlocked nodes also becoming deadlocked themselves. That is, once a queueing network falls into one of the transient deadlock states, it will eventually transition, either directly or through other transient deadlock states, into an absolute deadlock state.

Note that in [176] a classification of deadlock states are given which use similar vocabulary but different definitions to those given in this thesis. ‘Transient deadlock’ defined there corresponds to the blocking considered in this thesis, and isn’t considered a deadlock here. ‘Permanent deadlock’ there is equivalent to the deadlock that is discussed in this thesis, and encompasses both transient and absolute deadlock defined above. They further define ‘Total deadlock’, a subset of ‘Permanent deadlock' that corresponds to absolute deadlock defined here. Figure 2.7 illustrates these differences, blue text indicates terminology defined in this thesis, and red text indicates terminology defined in that paper; italicised text and shaded boxes indicate states considered as deadlock in each context. However, the authors of [176] state that there is no conceptual difference between transient and permanent deadlocks according to their definitions. Using Definition 2 of deadlock, we disagree, and their definition of transient deadlock is not considered a deadlock at all here. So this thesis considers a narrower definition of deadlock than considered in that paper.



(a) Deadlock states defined in [176]



(b) Deadlock states defined in this thesis.

Figure 2.7: Illustrating the types of deadlock, and the differences between the meaning of deadlock in this thesis and in some of the literature (adapted from [176]). Actual states are equivalent in both diagrams, only labelling differs.

This section has defined two types of deadlock observed in open restricted queueing networks: transient and absolute deadlock. In the next section, deadlock configurations will be discussed and enumerated.

2.4 Enumerating deadlock configurations

A deadlock configuration is a partition of the set of nodes of a queueing network into subsets of nodes in the same deadlock component, and the set of nodes that are not in deadlock. Let us define a deadlock component as follows:

Definition 7. *A deadlock component is a set of deadlocked nodes of a queueing network, such that the all customers at those nodes are blocked by one another.*

That is, a deadlock component is set of nodes that are in deadlock with one another. A queueing network may have many deadlock configurations, or states. These configurations, as shown in the previous section, may be transient or absolute.

This section enumerates the number of potential deadlock configurations in complete queueing networks.

Figure 2.8 illustrates the number of different types of deadlock configurations possible in a complete queueing network with four nodes. Each colour denotes a separate deadlock component, and the grey nodes show those nodes that are not in deadlock.

A Bell number B_n is defined as the number of ways a set of n elements can be partitioned into non-empty subsets [27]. Theorem 1 gives the number of possible deadlocked states in a complete queueing network.

Theorem 1. *If the queueing network Q with N nodes is complete, and every node has finite queueing capacity, then the number of possible deadlocked states is $B_{N+1} - 1$, where B_n denotes the nth Bell number.*

Proof.
- Define S_N as a set of N nodes.
- Define D_N as the set of possible deadlocks on a queueing network Q with N nodes.
- Define ξ as a dummy node.
- Define P_N as the set of all partitions of $S_N \cup \{\xi\}$ not including the set that counts all nodes in one partition.

By definition $|P_N| = B_{N+1} - 1$.

Consider the mapping $g : D_N \mapsto P_N$ defined in the following manner:

- Order the nodes of Q and S_N.

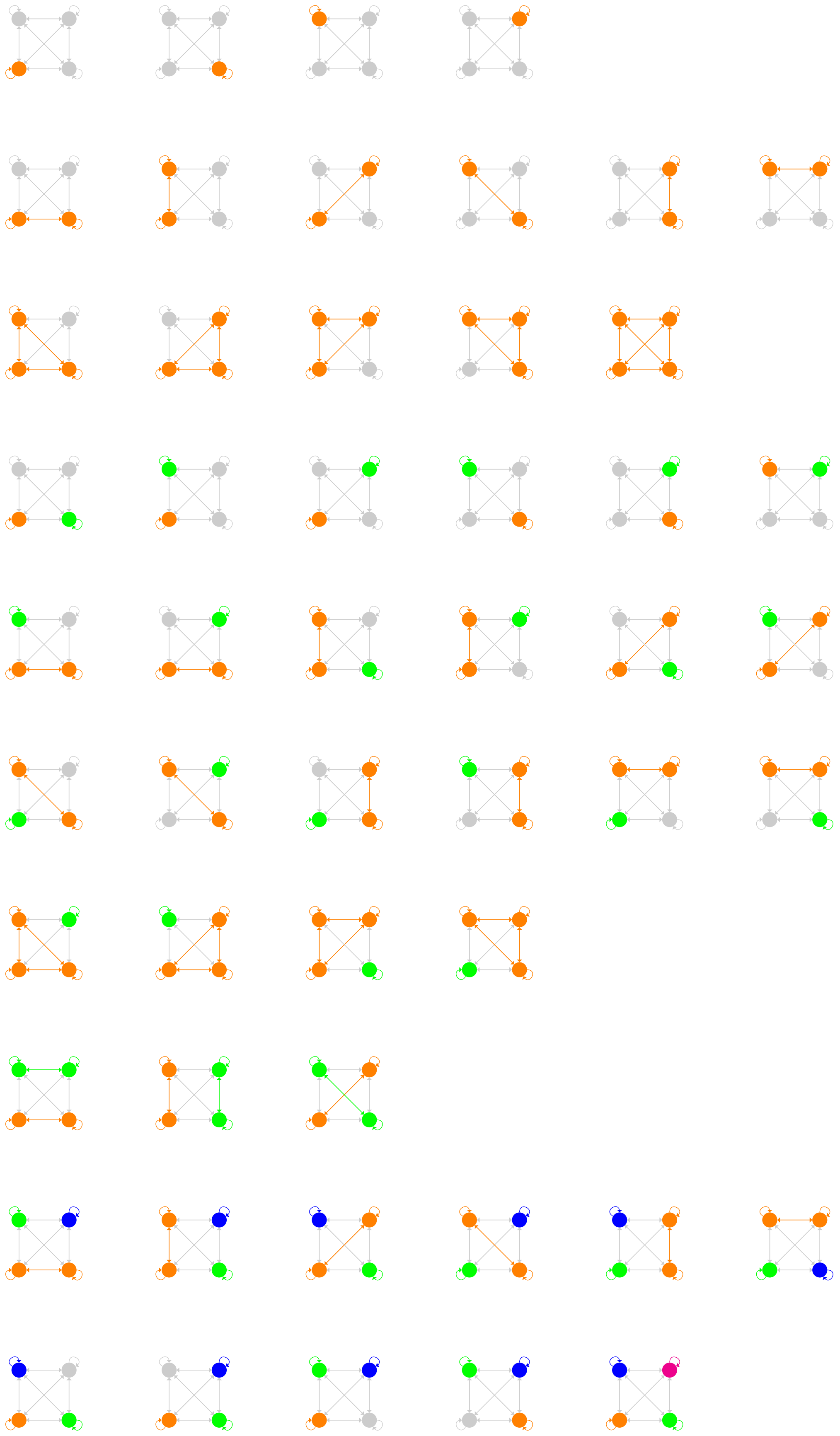


Figure 2.8: The $51 = B_5 - 1$ configurations of a 4 node complete queueing network.

- The node $x \in Q$ maps to the corresponding node $x' \in S_N$.
- Partition $S_N \cup \{\xi\}$ in such a way that:
  - For all $x \in Q$ that are in the same deadlock component, all corresponding $x' \in S_N$ are placed in the same partition of P_N.
  - For all $x \in Q$ that are not deadlocked, all corresponding $x' \in S_N$ are placed in the same partition as ξ.

Now as the network is complete, any set of nodes in Q may be in the same deadlock component as each other. In addition, a node may be in deadlock with itself, thus a deadlock component may contain only one node.

Consider the inverse mapping $g^{-1} : P_N \mapsto D_N$:

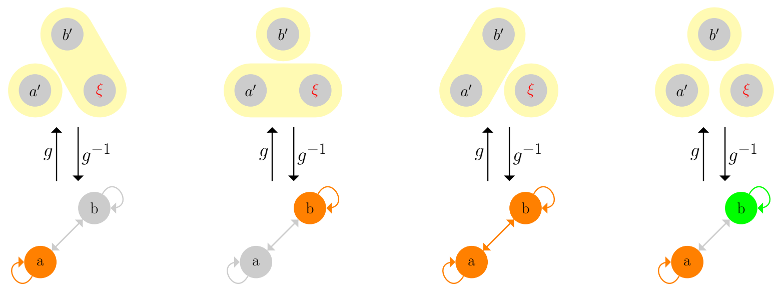
- Consider a set $Y \in P_N$:
  - If $\xi \in Y$ then all other elements of Y correspond to nodes in Q that are not deadlocked.
  - If $\xi \notin Y$ then all other elements of Y correspond to nodes in Q that are deadlocked together.

The mapping g is both 1-1 and onto, and so is a bijection. Therefore $|D_N| = |P_N| = B_{N+1} - 1$. $\square$

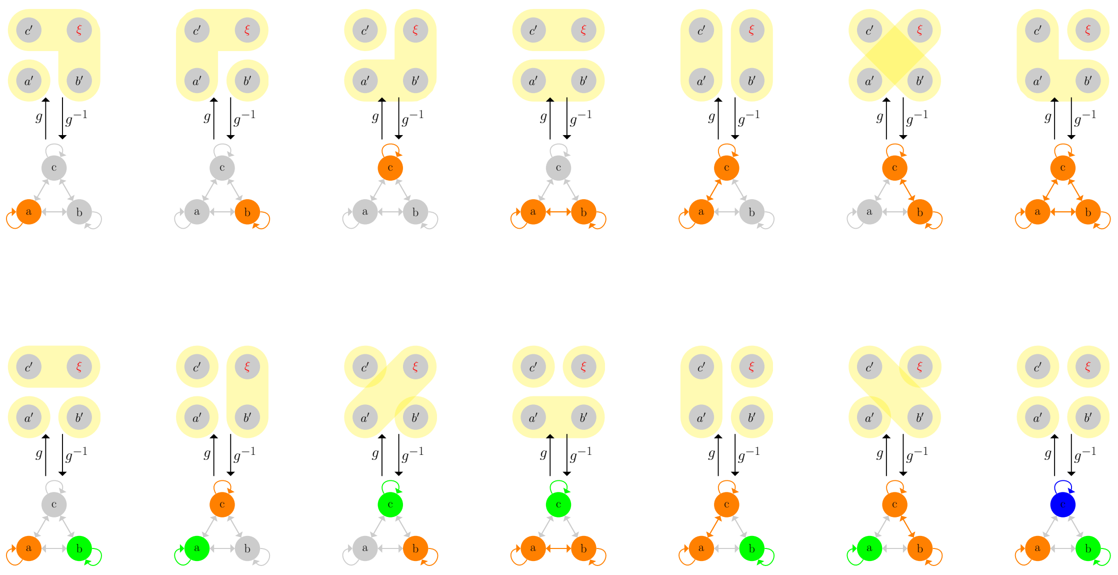
Figure 2.9 gives an example of the mapping g for $N = 2$ and $N = 3$.

Recall that each deadlock configuration can be classified as either being transient or absolute. Theorem 2 enumerates the number of potential transient and absolute deadlock configurations in a complete queueing network.

Theorem 2. *If the queueing network Q with N nodes is complete, and every node has finite queueing capacity, then the number of possible absolute deadlock states is B_N, and the number of possible transient deadlocked states is $B_{N+1} - B_N - 1$.*



(a) Example of g for $N = 2$



(b) Example of g for $N = 3$

Figure 2.9: Example of g for $N = 2$ and $N = 3$.

Proof. In an absolute deadlocked state every node is involved in some configuration of deadlock. The number of deadlock configurations where there are no non-deadlocked nodes is simply the number of partitions of N nodes, where each partition represents a separate deadlocked component. Therefore the number of absolute states is B_N from the definition of a Bell number.

A deadlocked state is either absolute or transient. If there are B_N absolute states, and $B_{N+1} - 1$ in total, then there must be $B_{N+1} - B_N - 1$ transient deadlocked states. □

Some deadlock configurations cannot be reached without first having gone through another deadlock state. For example, a deadlock configuration that consists of two separate components in deadlock must have been in a state with one deadlocked component prior to reaching that configuration.

The remainder of this chapter is only be concerned with the first instance of deadlock, that is the deadlocked states that can be reached from a non-deadlocked state directly. These are deadlock configurations with only one deadlocked component. Theorem 3 gives the number of different deadlocked states that can reached at first instance.

For this we will require the definition of Stirling numbers of the second kind $S(n, k)$ [27]. They give the number of ways to partition a set of n elements into k subsets.

Theorem 3. *If the queueing network Q with N nodes is complete, and every node has finite queueing capacity, then the number of possible deadlocked states that can be reached at first instance is $S(N + 1, 2)$, where $S(n, k)$ denotes a Stirling number of the second kind with parameters n and k.*

Proof.
- Define S_N, D_N, P_N, ξ and g as in Theorem 1.
- Define F_N as the set of possible deadlocks that can be reached at first instance on a queueing network Q with N nodes.

# Nodes (N)	# Deadlock configurations ($B_{N+1} - 1$)	# Transient configurations ($B_{N+1} - B_N - 1$)	# Absolute configurations (B_N)	# First instance configurations ($S(N+1, 2)$)
1	1	0	1	1
2	4	2	2	3
3	14	11	5	7
4	51	36	15	15
5	202	150	52	31
6	876	673	203	63
7	4139	3262	877	127
8	21146	17006	4140	255
9	115974	94827	21147	511
10	678570678569	678570562594	115975	1023
OEIS [164] number	A058692		A000110	A000225

Table 2.1: Summary of Theorems 1, 2, and 3, for complete queueing networks up to 10 nodes.

- Define T_N as the number of ways $S_N \cup \{\xi\}$ can be partitioned into two subsets.

Note that $F_N \subset D_N$ and $T_N \subset P_N$. By definition $|T_N| = S(N+1, 2)$.

Noting that a deadlocked state that can be reached at first instance only has one deadlocked component then the following must be true: if $y \in F_N$ then $g(y)$ is a partition of two subsets, one subset containing all the nodes in the sole deadlocked component, and one subset containing all non-deadlocked nodes.

Thus g maps $F_N \to T_N$. Recall that g is both 1-1 and onto, thus $|F_N| = |T_N| = S(N+1, 2)$. □

This section has presented three results on the enumeration of deadlock configurations in complete queueing networks: the number of deadlock configurations, the number of transient and absolute deadlock configurations, and the number of deadlock configurations that can be reached at first instance. Table 2.1 summarises these results for complete queueing networks up to 10 nodes. As the number of nodes increase, the number of deadlock configurations quickly increases. The bottom row gives the corresponding OEIS [164] sequence number for that column.

2.5 Detecting deadlock

In the following subsections, a method will be presented to detect deadlock in discrete event simulations of queueing networks.

2.5.1 The state digraph

A method for detecting when deadlock occurs in an open queueing network Q with N nodes, using a dynamic directed graph as a representation of the system state, the state digraph, is now proposed. In previous literature on wait-for graphs these are bespoke graphs that represent system states, where edges denote some form of waiting or blockage relationships. Here we present a generic state digraph that is defined for all FIFO queueing networks that exhibit Type I blocking:

Let the number of servers in node i be denoted by c_i. Define $D(t) = (V, E(t))$ as the state digraph of Q at time t.

The vertices V correspond to servers in the queueing system, and is static over time. Thus, $|V| = \sum_{i=1}^{N} c_i$.

The edges at time t, $E(t)$ correspond to a blockage relationship. There is a directed edge at time t from vertex $X_a \in V(t)$ to vertex $X_b \in V(t)$ if and only if an individual occupying the server corresponding to vertex X_a is being blocked by an individual occupying the server corresponding to vertex X_b.

This leads us to the following formal definition:

Definition 8. *The state digraph $D(t)$ of a queueing network is defined by that network's state at any time t. Vertices of the state digraph correspond to servers of the network. A directed edge denotes a blockage relationship in the following manner: if a customer at the kth server of node i is blocked from entering node j, then there are directed edges from the vertex corresponding to node i's kth server to* *every vertex corresponding to the servers of node j.*

The state digraph $D(t)$ can be partitioned into N service-centre subgraphs, $D(t) = \bigcup_{i=1}^{N} d_i(t)$, where the vertices of $d_i(t)$ represent the servers of node i. The vertex set of each subgraph is static over time, however their edge sets change.

The state graph is dynamically built up as follows, adhering the rules set out in Definition 8.

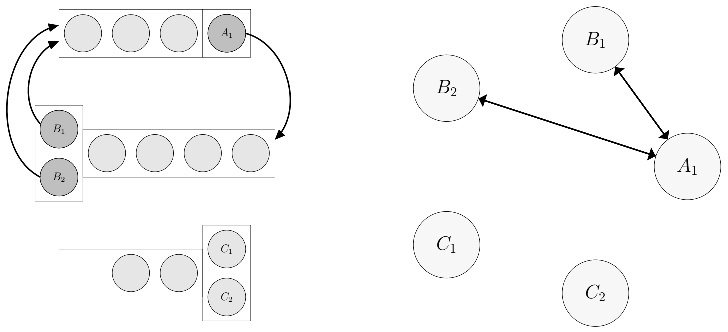
- When an individual finishes service at node i, and this individual's next destination is node j, but there is not enough queueing capacity for j to accept that individual, then that individual remains at node i and becomes blocked. At this point c_j directed edges between the vertex corresponding to this individual's server and the vertices of $d_j(t)$ are created in $D(t)$.
- When an individual is released that servers out-edges are removed.

Figure 2.10 shows examples of queueing networks, both in and out of deadlock, and the corresponding state digraph in each case. Note the directions of the arrows, some are double-ended if there exists an edge in $D(t)$ going in both directions.

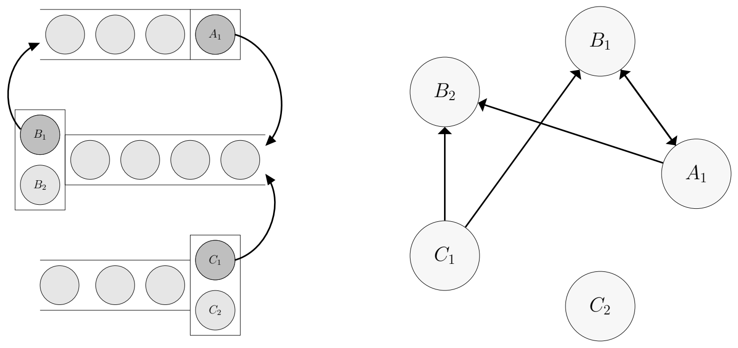
2.5.2 Properties of the state digraph

For this sections, recall the following graph theoretic definitions [16, 50, 187]:

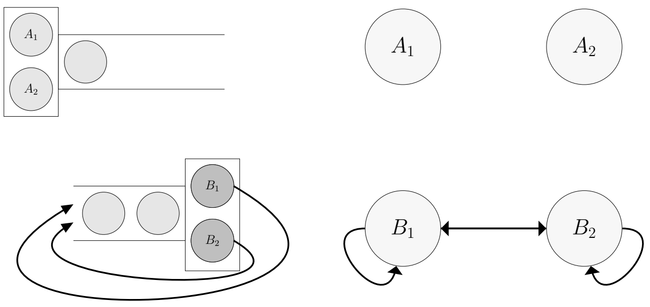
- **Order**: The order of the directed graph D is its number of vertices, denoted by $|V(D)|$.
- **Weakly connected component**: A weakly connected component of a digraph containing X is the set of all nodes that can be reached from X if we ignore the direction of the edges.
- **Direct successor**: If a directed graph contains an edge from X_i to X_j , then we say that X_j is a direct successor of X_i.



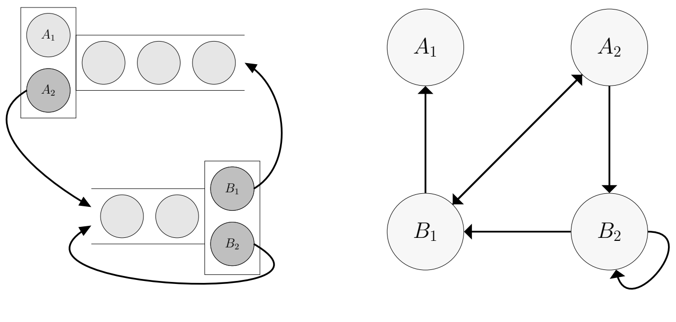
(a) A three node queueing network in deadlock, with state digraph.



(b) A three node queueing network not in deadlock, with state digraph.



(c) A two node queueing network in deadlock, with state digraph.



(d) A two node queueing network not in deadlock, with state digraph.

Figure 2.10: Example state digraphs with their corresponding queueing systems.

- **Ancestors**: If a directed graph contains a path from X_i to X_j , then we say that X_i is an ancestor of X_j.
- **Descendants**: If a directed graph contains a path from X_i to X_j , then we say that X_j is a descendant of X_i.
- $\mathbf{deg^{out}}(X)$: The out-degree of X is the number of outgoing edges emanating from that vertex.
- **Subgraph**: A subgraph H of a graph G is a graph whose vertices are a subset of the vertex set of G, and whose edges are a subset of the edge set of G.
- **Sink vertex**: A sink vertex is a vertex in a directed graph that has out-degree of zero.
- **Knot**: In a directed graph, a knot is a set of vertices with out-edges that are inescapable by traversing the edges.

Now consider the state digraph $D(t)$. Consider a vertex v in $D(t)$. Some observations:

- If the server corresponding to v is unoccupied, then v has no incident edges.
- It can be interpreted that all vertices with a path to v correspond to servers whose individuals are being blocked directly or indirectly by the customer at the server corresponding to v.
- Similarly it can be interpreted that all vertices that v has a path to correspond to servers whose occupants are directly or indirectly blocking the customer at the server corresponding to v.
- It is clear that if all vertices that v has a path to correspond to servers occupied by blocked individuals, then the system is in deadlock at time t.

Results are now presented that can be used to detect deadlock for open restricted queueing networks.

Theorem 4. *A deadlocked state arises at time t if and only if $D(t)$ contains a knot.*

Proof. Consider a queueing network with set of servers S. Consider the state digraph $D(t) = (V, E(t))$. Note that there is a 1-1 pairing between the elements of S and the elements of V, by Definition 8 of $D(t)$.

- Assume the system is in deadlock at time t. By Definition 2 (of deadlock) there exists $\mathcal{S} \subseteq S$ a subset of servers with blocked customers blocked only by customers at servers in $\mathcal{S}$. Consider $\mathcal{V} \subseteq V$ corresponding to $\mathcal{S}$.

  For each $s \in \mathcal{S}$ the corresponding $v \in \mathcal{V}$ has at least one out-edge in $E(t)$ because the customer at s is blocked (by Definition 8).

  By Definition 2 every $v \in \mathcal{V}$ has at least one path in $E(t)$ to a vertex in $\mathcal{V}$, and has no path in $E(t)$ to any vertex outside of $\mathcal{V}$.

  By definition of a knot, there exists $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E} \subseteq E(t)$, such that G is either a knot or a collection of knots.

- Assume that $D(t)$ contains a knot $G = (\mathcal{V}, \mathcal{E})$. Consider $\mathcal{S} \subseteq S$ corresponding to $\mathcal{V}$.

  As G is a knot, every $v \in \mathcal{V}$ has an out-edge, thus every customer at $s \in \mathcal{S}$ is blocked (by Definition 8).

  As G is a knot, there is no path in $E(t)$ from any $v \in \mathcal{V}$ to any vertex outside of $\mathcal{V}$. Therefore every customer at $s \in \mathcal{S}$ is blocked directly or indirectly by customers at servers in $\mathcal{S}$.

  By Definition 2 this implies the system is in deadlock at time t.

□

The knot condition can be simplified for specific cases. Theorems 5 and 6 may offer computational advantages in cases where it is easier to identify weakly connected components or sinks than it is to identify knots.

Theorem 5. *For queueing networks:*

1. *with one node*
2. *with two nodes, each with two or fewer parallel servers*
3. *with a finite amount of nodes, each with a single-server*

a deadlocked state arises at time t if and only if $D(t)$ contains a weakly connected component without a sink.

Proof. To prove the result in one direction, each case is considered separately:

1. Consider a one node queueing network.

   If there is deadlock, then all servers are occupied by blocked individuals, and so all vertices corresponding to those servers have an out-edge. Thus there are no sinks.

2. Consider a two node queueing network, each node with 2 or fewer parallel servers.

   If both nodes are involved in the deadlock, so there is at least one customer in node 1 blocked from entering node 2, and at least one customer from node 2 blocked from entering node 1, then all servers in node 1 and node 2 in $D(t)$ will have out edges as they are occupied by a blocked individual. The servers of node 1 and 2 consist of the entirety of $D(t)$, and so there is no sink nodes.

   Now consider the case when only one node is involved in the deadlock. Without loss of generality, consider that node 1 is in deadlock with itself, then the servers of node 1 have out-edges. For the servers of node 2 to be part of that weakly connected component, there either needs to be an edge from a server in node 1 to a server in node 2, or an edge from a server in node 2 to a server in node 1. An edge from a server in node 1 to a server in node 2 implies that a customer from node 1 is blocked from entering node 2, and so node 1 is not in deadlock with itself. An edge from a server in node 2 to a server in node 1 implies that a customer in node 2 is blocked from entering node 1. In this case one server in node 2 has an out-edge. Now either the other server of node 2 is empty or still in service, and so isn't part of that weakly connected component, or the other server's customer is blocked and so has an out edge. Thus there are no sinks.

3. Consider a queueing network with N nodes, each with a single-server.

   If n $(1 \leq n \leq N)$ nodes are involved in the deadlock, then each server in those n nodes has a blocked customer, and so the corresponding vertex in the state digraph has an out-edge.

   Of the nodes not involved in that deadlock, the vertices corresponding to their servers can only be in the same weakly connected component if:

   - They contain a blocked individual that is blocked to the nodes involved in the deadlock.
   - Individuals in the deadlocked nodes are blocked to those nodes not involved in the deadlock.

   In the first case, the vertices corresponding to the servers in those nodes will all have an out-edge. In the second case it is implied that the customers at servers in deadlocked nodes are blocked to both a node in deadlock and a node not in deadlock, which is not possible (customers can only be blocked to one location at a time).

   Thus there are no sinks.

Proving the result in the other direction is equivalent to proving that a weakly connected component without a sink contains a knot:

- Consider a weakly connected component, G, of $D(t)$.
- Assume G contains no knots. By definition of a knot, this implies:
  - G contains a sink; or

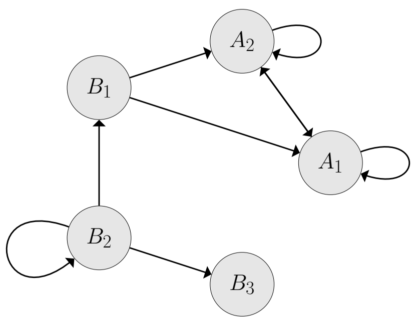


Figure 2.11: State digraph of the counter-example.

- G contains a vertex with a path to another vertex outside of G (contradicting the fact that G is a weakly connected component).

□

In the general case using the result of Theorem 5 is not sufficient to detect deadlock. In order to illustrate this, consider the following counter-example of a two node queueing network, where node A has two servers, node B has three servers. Beginning with all servers occupied by customers in service and full queues. The customer at server A_1 becomes blocked to node A. The customer at server B_1 becomes blocked to node A. The customer at server B_2 becomes blocked to node B. The customer at server A_2 becomes blocked to node A. Node A is now in deadlock. The resulting state digraph, shown in Figure 2.11, has a weakly connected component with a sink.

If detecting absolute deadlocks only, then the condition further simplifies:

Theorem 6. *An absolute deadlocked state arises at time t if $D(t)$ doesn't contain a sink vertex.*

Proof. A vertex with out-degree greater than zero represents an occupied server whose occupant has finished service and is blocked. If all vertices have out-degree greater than zero, then all servers are occupied by blocked individuals. A release at vertex X_a can only be triggered by one of X_a's descendants finishing service. As all servers are occupied by blocked individuals, no server can finish service, and so no server can release their occupant, implying an absolute deadlocked state. □

Details on how the state digraph method of detecting deadlock is implemented into a simulation framework are given in Chapter 4, Section 4.5.

2.6 Summary

This chapter has defined and analysed deadlock in open restricted queueing networks. In queueing systems these have been categorised into transient and absolute deadlock. In transient deadlock only a part of the system has permanently ceased movement due to mutual blocking, while there are other customers in the system free to continue as usual. On the other hand in absolute deadlock all customers have permanently ceased all movement. Furthermore 'first instance' deadlocks are defined, those deadlock states that can be reached from a non-deadlocked state.

Enumerations of numbers of possible deadlocks of each of these type have been given for complete queueing networks. For a complete queueing network with N nodes, there are $B_{N+1} - 1$ deadlock configurations; $B_{N+1} - B_N - 1$ of those are transient, and B_N are absolute. $S(N+1, 2)$ of these configurations can be reached at first instance.

Finally, the state digraph, a graph theoretic representation of a state of a queueing network, has been defined. Using this construction, results have been presented that detect deadlock in discrete event simulations of queueing networks. In general a knot in the state digraph indicates that deadlock has been reached. For some systems, namely one node systems, two node systems with two or fewer servers, and single server systems, this condition is simplified to only require a weakly connected component without a sink. Moreover if the state digraph does not contain a sink, then an absolute deadlock has been reached.

Chapter 3

An Open Source Simulation Framework

Discrete event simulation (DES) is an extremely popular and rapidly growing method of analysing networks of queues [23, 61, 146], and is a standard approach to studying a variety of real life operational systems. It is one of the most widely used techniques in operational research and management science [92]. It allows for artificially experimenting with a system and determining its behaviour without costly and laborious modifications to the real physical system.

A definition of the kind of simulation that is discussed here is given in [147]:

> *"Experimentation with a simplified imitation (on a computer) of an operations system as it progresses through time, for the purpose of better understanding and/or improving that system".*

In general the operations system here refers to a collection of parts organised for some purpose. For a queueing system that includes customers, service centres and servers.

This chapter introduces the Ciw Python library for conducting discrete event simulations. Ciw was developed in order to enable reproducibility and replicability of discrete event simulations, while encouraging best practices. It is an open-source Python library for the simulation of open queueing networks using the event scheduling simulation approach.

This chapter is structured as follows:

- Section 3.1 discusses the need for simulation software that enables reproducible research, and identifies key characteristics required from such software.
- Section 3.2 outlines important steps in the software development process that ensure sustainable and trustworthy software.
- Section 3.3 gives examples of the library's usage, and showcases some of its features.
- Section 3.4 compares Ciw with other popular discrete event simulation tools in terms of both performance and their capabilities for reproducible research.
- Section 3.5 summarises the contents of this chapter.

The work in Sections 3.1, 3.4 of this chapter have been published in [124].

The source code for all versions of Ciw can be found on GitHub [53] at `https://github.com/CiwPython/Ciw`, are hosted on PyPI [167] at `https://pypi.python.org/pypi/Ciw/`, and are archived on Zenodo at `https://doi.org/10.5281/zenodo.1145638`. Its documentation is found at `http://ciw.readthedocs.io`. The library was built as part of this PhD with code contributions by others (listed at `https://github.com/CiwPython/Ciw/blob/master/AUTHORS.rst`), and further support by many more including colleagues and users.

Note that throughout this chapter the term *date* will be used to denote a specific point in time, whereas *time* will be used to describe a length of time. General time units are used that can later be transformed into specific units such as hours or days. When writing code blocks, any line beginning with `>>>` denotes code that is actually run.

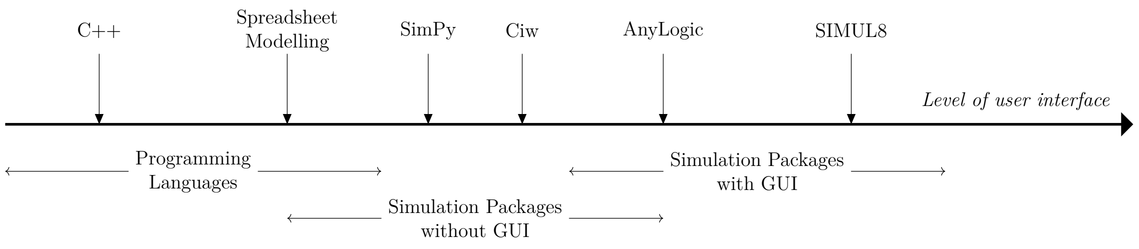


Figure 3.1: A suggested spectrum corresponding to user interface, with illustrative positioning for six simulation options.

3.1 Motivation

Reproducibility and replicability, described as "the cornerstone of cumulative science" [149], is critical in order to assert correct results and build on the work of others [67, 110, 149]. In computational research this can be achieved by following a number of best practices [1, 22, 33, 67, 75, 136, 149, 186]. In simulation modelling, it is a known issue [110].

A popular software paradigm for research is open source software which implies software source code that is freely usable and modifiable. In a recent review of open source discrete event simulation software [35], 44 open source discrete event simulation solutions were found and reviewed, however not all followed best practice: 14 were found to have no available documentation, and over half failed to use any version control.

Simulation options traditionally fall into discrete categories [92, 147], consisting of programming languages, simulation packages, and spreadsheet modelling. Simulation frameworks can be considered to be on a spectrum corresponding to the user interface. Such a spectrum is shown in Figure 3.1, with some suggested positions for a selection of simulation options, including Ciw, SimPy [162], AnyLogic [163], SIMUL8 [153], as well as building a simulation in C++, and spreadsheet modelling. Note that spreadsheet modelling, despite usually being interfaced with a graphical user interface, is considered here a type of programming language due to its generic nature and syntax.

Advantages and disadvantages of these methods have been discussed extensively [20, 35, 92, 147]. Programming from scratch is considered more flexible, is bespoke, may improve speed, and increases the variety of performance measures collected. However, a lack of user interface may hinder model communication. It is discussed in some literature that simulation packages, especially those with a graphical user interface (GUI), are more accessible, easily modifiable, and easier to communicate with non-specialists. GUIs can aid with conceptual modelling, model validation and verification [20, 21, 84].

However some simulation packages come with several disadvantages for example high costs (licences, training, plug-ins and maintenance), they often lack modularity, low model reusability, and lack of access to the source code can impede understanding, customisation and flexibility. Furthermore, it is suggested in [20] that the addition of a GUI can lead to bad simulation practice. This includes: disregarding formal methodology such as statistical analysis in favour of watching animations; introducing bias in the model building process by building models that represent how a system should work instead of how they actually work; and false model validation in which realistic graphics imply a realistic model.

Two themes arise when discussing research and software development: reproducibility and sustainability, and best practice [1, 22, 33, 67, 75, 136, 149, 186]. The Ciw library aims to enable users to meet these standards in the domain of discrete event simulation.

Guidelines for the reporting of simulation models to ensure reproducibility are given in [110], including reporting on code access and the mechanics of model implementation. Open and readable simulation models can facilitate this. In [81] three properties are listed as minimum requirements to ensure reproducibility in simulation:

- Readability
- Modularity

- Extendibility

All three properties can apply to Python [169], the ecosystem in which modelling with Ciw takes place. Python is an open source, object-orientated, high level language. Python's advantages as a language for developing simulation models are listed in [35]. These include its intuitive and readable syntax, and potential to form a community of users and developers. In addition, Python is attractive to researchers due to the extensive collection of other scientific libraries available to integrate work, for example combining simulation with data analysis, machine learning, and optimisation. An example of this as a teaching exercise with Ciw is seen in [85]. There are a number of popular scientific Python libraries, including:

- NumPy [181] and SciPy [76] for scientific computation;
- scikit-learn [128] for machine learning and optimisation algorithms;
- Pandas [104] for data analysis tools;
- Matplotlib [69] and Seaborn [182] for data visualisation;
- SymPy [107] for symbolic mathematics.

In [186] the authors discuss the advantages of using high level programming languages such as Python for research software over low level languages like C and Fortran. Advantages include an increase in productivity when writing in high level languages, better readability, and rapid design decisions and prototyping. A downside however, due to the fact that Python is an interpreted language, is that the computational speed will not be as fast as compiled languages.

Another strength of Ciw is that it is completely open source. It has one of the most flexible and permissive licences, the MIT licence, and is written in an open source language. This offers the user immediate advantages over commercial-off-the-shelf simulation packages. All source code is available for inspection, testing, and modification. This enables and encourages greater understanding of the underlying methodology, increases model confidence, and provides an extendible framework in which discrete event simulation may be carried out. Furthermore, the elimination of license fees and maintenance costs facilitate model sharing, open science, and reproducibility. This overcomes common problems with commercial software with more stringent licences, where models may sometimes not be shared between two computers with the same software.

This ecosystem provides an opportune way in which reproducible scientific research can be conducted:

- All manual data manipulation can be avoided [33, 149, 186];
- All raw data can be saved [149];
- All models can be version controlled [22, 149, 186];
- All models can be scrutinised by automated testing [22, 186];
- All models can be shared [22, 33, 67, 75, 149].

Ciw is also developed in a sustainable manner, and strives to follow best practice in research software development. This includes extensive testing (it has 100% test coverage [18]), comprehensive documentation, readability, modularisation, transparency, and use of version control [136, 186].

Object-orientation, an important feature of Python, lends itself well to simulation [35, 92]. In [35] the authors state that "DES is a traditional paradigm where object orientation is intuitively adopted". The argument for linking object-orientation to one particular method of discrete event simulation, the three-phase approach, is given in [134]. Breaking a simulation down into events, activities and entities, as is required for the three-phase approach, is a form of modularisation itself. This equally applies to the similar event scheduling approach used in Ciw. In addition, simulation modellers habitually think of entities as belonging to a class, or classes, of similar entities. It is intuitive to build systems like this in an environment where modularisation is key, such as in an object-orientated programming language. Further advantages of using object-orientation are listed in [92]: its flexibility, its ability to deal with complexity through modularisation, and its high reusability.

As stated previously, using open source software provides distinct advantages over traditional commercial-off-the-shelf simulation packages. Similarly, open source development can provide many advantages over closed source development. However, [35] argues that apart from eliminating the licence fee, simply being open source does not offer immediate advantages for developers, but it is the ethos and culture that comes with open source that provide the advantages. It is argued in [180] that open source culture provides incentives to innovate, as there is no need for a large demand or promise of recoupment of financial investment for certain features to be developed. That is, private needs create public goods. This has been evident in Ciw, where new research can be directly implemented into the software and tested and experimented quickly. New features have been implemented after discussions with users from around the world via the online issue tracker. Freedom of development is another crucial aspect of open source according to [180], where users can fork and develop their own versions of software for their bespoke needs. An argument for promoting best practice in open source software is given in [1], as it achieves better quality software. Some of these best practices arise naturally in an open source environment, for example rapid release cycles, code reviews, and code modularity. Open source development actually encourages these best practices due to its transparency and the opportunities for developers to showcase their work [75]. This ties in with the “Release early, release often” [140] mantra that is often used to encourage early imperfect releases that other developers may test and contribute to; and to encourage regular releases so that bugs are fixed early and there isn’t too much of a jump in usability between versions.

A review of open source discrete event simulation software is given in [35], and as mentioned previously, a number of frameworks failed to follow best practice in their development. Three Python libraries were found, though only one was found to meet the quality requirements of the study, SimPy [162]. Further, [81] draws many parallels between open source development and simulation modelling, while concluding that the "steady, long-term progress toward libraries of easily extendible and easily reusable simulation code" is an important direction for simulation modellers.

Thus Ciw is developed as an object-orientated, open source Python library with the following qualities which aim to encourage and enable reproducible simulation research:

- Open, accessible source code promotes understanding, development and modification. Online issue tracker and open development environment fosters discussion, idea generation, and development. Permissive licence allows it to be extended, modified and shaped to the users' needs.
- Code development follows best practice guidelines for reproducibility, code quality, and modification. Modularity allows modification and extension through inheritance.
- Python ecosystem allows it to be used flexibly within the programming language, allowing ease of experimentation and integration with other scientific tools. Models can be tested and version controlled.
- Models are readable and the package has extensive bilingual documentation, to enhance model communication.

3.2 The software development process

Software is an important scientific tool, with many researchers writing their own software. This is often a necessity as ample domain-specific knowledge is required in its design and development. This was prevalent when developing Ciw. As discussed in Section 3.1, two themes that arise as vital in research software development are sustainability and best practice. For best practices when developing research software comprehensive guidance in eight areas is given in [186]:

- **Write programs for people, not computers:** Code should be written to be read by humans, this includes other developers and potential collaborators. This ensures that there is no one person who alone can understand the methods used, code written, and usage of the software, so that research and reproducibility does not die with that person. Some ways of achieving this include writing consistent and meaningful variable names, and simplifying code such that readers need only keep the minimum amount of facts in their memory.

- **Let the computer do the work:** Any repetitions of tasks should be automated. This not only wastes time, but reduces the number of inevitable mistakes made during the development of the code. In order to do this, any task that is required to produce output results should be automated, and thus can be run any time any changes to the data/inputs/code occur.

- **Make incremental changes:** Due to the nature of scientific research, where results often dictate future research directions, the requirements of the next version of software often rely on results of previous versions. By making small incremental changes to software, researchers can keep on top of changing requirements. Agile development [65] is a set of methods that can achieve this. In general their principles focus on:

  – Working code. Implementing a small change resulting in working code is better than promising big changes where working code may be some time away.

  – Using people effectively. People transfer ideas faster face to face than by writing documents. Few designers working together can produce better designs that each could alone.

  – Response to change. Change is encouraged in the context of constant feedback and discussions.

Keeping track of changes is vital, and being able to do so in a collaborative setting has posed challenges. Version control is the standard solution, and is discussed more in Section 3.2.1.

- **Don't repeat yourself (or others):** Code should be modularised, rather than copy and pasting large chunks of code for different purposes. Code modularisation also makes re-purposing the code for other uses easier. Developers should not repeat the work of others, that is writing their own solutions to problems that have been solved with other software (e.g. matrix inversion).

- **Plan for mistakes:** Mistakes are inevitable, however they can be reduced by asserting and testing the software. This is discussed in Section 3.2.2.

- **Optimise software only after it works correctly:** Writing software in fast low-level languages slow down the development process and may discourage change and improvement. Writing in the highest level language possible increases productivity. Although the code may not yet be optimised, researchers can write more code in less time.

- **Document design and purpose, not mechanics:** Good documentation helps people understand code and enables reproduction of results. This is discussed in Section 3.2.3.

- **Collaborate:** Just as peer reviewed manuscripts help improve research and reduce errors, collaboration and code reviews help eliminate bugs, enhance readability, and improve the code. Many tools help ensure best practice when collaborating with others, including pre-merge code reviews, and issue trackers.

The next few subsections will outline some methods of best practice considered and implemented when developing Ciw.

3.2.1 Version control & collaboration

A version control system records all changes to a file repository over time, tracking its development. It also provides the opportunity to recall previous versions of files.

This type of system is essential for ensuring reproducibility of scientific research [149, 186]. Even small changes in the computer program may have unindented consequences in the output or results of the code. Recording the exact state of the code that was used to produce scientific results is crucial in order to allow others to reproduce and verify the results.

Common features of version control systems are listed in [148]:

- **Backup and restore:** Files can be saved as they are edited, while all previous versions can be accessed and restored.
- **Synchronisation:** Source code files can be shared, and each user's codebase can be updated with the latest version.
- **Undo changes:** Any changes made are not permanent as previous versions can be easily accessed.
- **Track changes:** Messages are attached to file changes in order to track why and how those changes were made, and how the code evolves over time.
- **Track ownership:** All changes are tagged with the person who made that change.
- **Sandboxing:** The ability to make changes in an isolated area that will not affect the rest of the code.
- **Branching and merging:** Code can be copied and then modified in isolation, before being merged back intro the original code.

There are a number of popular tools for version control, these include Git [52], Subversion [9], and Mercurial [106]. Many version control systems record differences in files. Git however, records snapshots of file systems [28], and envisions data as a stream of snapshots. Versions are then stored as references to those snapshots. Git was chosen as the version control tool for the development of Ciw.

Making the source code for computer programmes publicly available on-line is another very important step in reproducibility [67, 136, 149]. This aids in the accessibility of reproducing work, making it easier and simpler for other researchers to verify scientific work. Another advantage of hosting code publicly, along with all its version control information, is transparency. This allows many different people to view the code during its development, offering different perspectives, raising issues, finding bugs, and contributing or requesting features. Services such as GitHub [53], SourceForge [155], Gitlab [54], and BitBucket [10] allow users to host code repositories publicly. GitHub was chosen as the host of Ciw's source code. This allows all code, supplementary material, and version control history to be made available on a cloud service that integrates well with the version control system Git.

An important feature of GitHub is that it fosters collaboration between users. First it is a social service, allowing users to comment and raise issues on each other's repositories, and ask for help or assistance. More importantly, it encourages code contributions through its fork and pull request features. Here users can fork, or copy, another user's repository, make modifications on that fork, and then send a request to the original repository asking for the new contributions to be accepted. In this manner users automatically get credit for their contributions, cultivating a community of developers that support one another through contributions.

Figure 3.2 show an example of the GitHub contributions made to the Ciw library.

3.2.2 Automated testing

Testing code is of paramount importance in order to ensure reliability and robustness to new features and re-factoring. The standard method of testing code is through

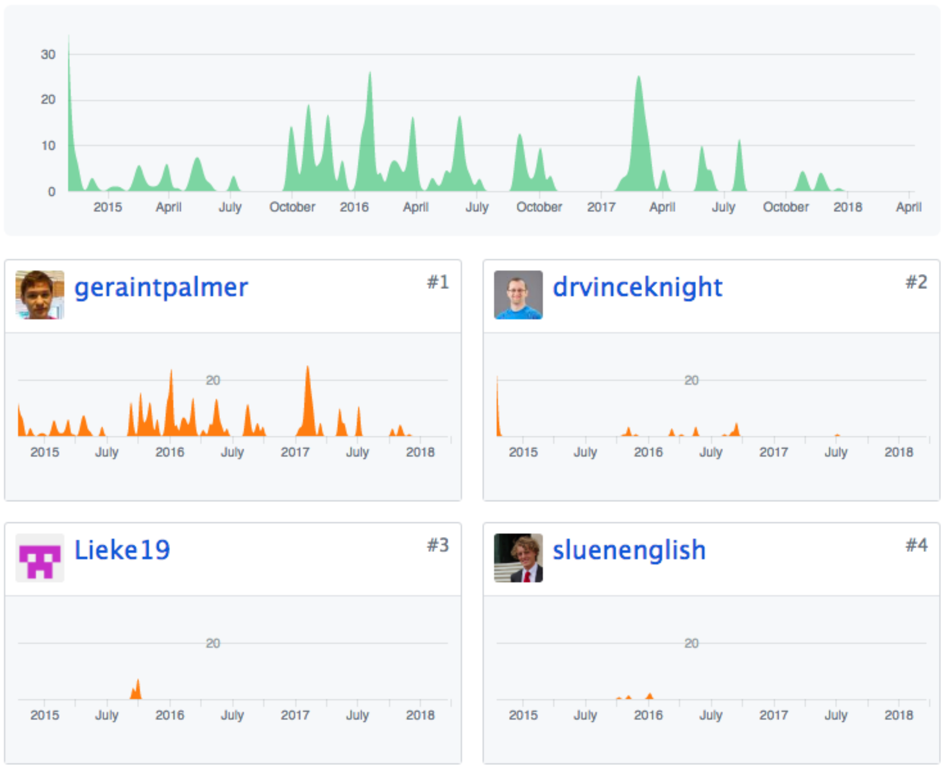


Figure 3.2: GitHub contributions made to the Ciw library.

automated testing using test suits that run parts of the code and assert whether they are behaving as expected.

Two types of tests are described in [131], functional tests that assert the code's functionality, and unit tests that help ensure the code is clean and free of bugs.

The main purpose of functional tests is to test how the whole application functions. They test the application from the perspective of the outside user, feeding in basic inputs and testing weather the end product or final behaviour is as expected. They are also called 'acceptance tests', 'end-to-end tests', or 'black box tests'.

Two types of unit test are described: 'pure' or 'isolated' unit tests, and 'integrated' tests. Unit tests assert that small chunks of code behave as expected, and test the application from the inside, the point of view of the programmer.

Pure unit tests are isolated from the rest of the code. A well written unit test only tests one function or method, or even one part of one function or method. If this well written unit test fails, it should be due to problems with that part of the code only, and not any other bit of code. Pure unit tests are fast and readable, however they do not test how well functions and methods integrate with one another. In order to test these, unit tests must be written that rely on other parts of the code that aren't explicitly being tested. This type of test is called 'integrated tests'.

In order to test an application fully, functional, integrated and pure unit tests are usually used in combination.

There are a number of tools available to aid the automated testing process. Some of these that are relevant to Python include the `unittest` library, coverage, coveralls, hypothesis, and Travis CI. These will be discussed in the following subsections.

3.2.2.1 The `unittest` library

The `unittest` library is one of Python's standard modules, and it is through this library that many testing suites are written. There are a number of popular alternatives [168] including `nose` and `pytest`.

Tests written in this library are structured in a way such that they *assert* whether the output given by one statement is equal to the output of another statement. Examples of writing tests are given in Figure 3.3, for a function that adds two numbers. First a concrete example is given, that $2 + 3 = 5$. Next two tests are given that ensure that the commutative property of addition holds $2 + 3 = 3 + 2$. The final test ensures that an error is raised if the wrong variable type is given to the function, that is a string and an integer cannot be added together.

3.2.2.2 Test coverage

The coverage tool [18] checks how much of the code is covered by the testing suite. When tests are run through coverage, every line of the module that has been run is recorded. Any unrecorded lines are flagged as being not covered by the testing suite.

```python
import unittest

def add(a, b):
    return a + b

class TestExample(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(2, 3), add(3, 2))
        self.assertTrue(add(2, 3) == add(3, 2))
        self.assertRaises(TypeError, add('two', 3))
```

Figure 3.3: Example of using the `unittest` library.

An example of a coverage report is shown in Figure 3.4. The 'Missing' column flags the lines of the code that haven't been run in the tests, making this an informative tool to aid good testing.

The performance measure that coverage outputs is the coverage percentage, with users striving to reach 100% coverage. Testing every function or method, even what may seem like trivial lines of code is still very important [131]. Not only does this provide rigour and completeness to the code, but the tests can also act as a place-holder in order to be prepared for when the code eventually grows in complexity. By checking coverage, we ensure that we are testing every case, even extreme cases, for example by ensuring coverage of every case of an if-statement, however improbable that may be in practice.

Nonetheless, simply gaining 100% coverage still does not guarantee fully tested and reliable code. One carefully selected functional test may in fact hit every line and produce the expected output, but this does not certify that the code works as expected for all cases.

3.2.2.3 Property based testing

Example based testing which has been discussed in prior sections has the flaw that it cannot test the code on all possible input data [47]. Usually only a handful

```
$ coverage run --source=ciw -m unittest discover ciw.tests
$ coverage report -m
Name                                 Stmts   Miss  Cover   Missing
---------------------------------------------------------------
ciw/__init__.py                         14      0   100%
ciw/arrival_node.py                     73      0   100%
ciw/auxiliary.py                        20      0   100%
ciw/data_record.py                       2      0   100%
ciw/deadlock_detector.py                46      0   100%
ciw/exactnode.py                        22      0   100%
ciw/exit_node.py                        15      0   100%
ciw/import_params.py                   189      0   100%
ciw/individual.py                       21      0   100%
ciw/network.py                          23      0   100%
ciw/node.py                            265      0   100%
ciw/server.py                           18      0   100%
ciw/simulation.py                      234      0   100%
ciw/state_tracker.py                    58      0   100%
ciw/tests/__init__.py                    0      0   100%
ciw/tests/test_arrival_node.py         228      0   100%
ciw/tests/test_auxiliary.py             65      0   100%
ciw/tests/test_data_record.py           55      0   100%
ciw/tests/test_exit_node.py             39      0   100%
ciw/tests/test_individual.py            81      0   100%
ciw/tests/test_network.py              365      0   100%
ciw/tests/test_node.py                 431      0   100%
ciw/tests/test_sampling.py             410      0   100%
ciw/tests/test_scheduling.py           175      0   100%
ciw/tests/test_server.py                55      0   100%
ciw/tests/test_simulation.py           441      0   100%
ciw/tests/test_state_tracker.py        195      0   100%
ciw/tests/test_version.py                5      0   100%
ciw/version.py                           1      0   100%
---------------------------------------------------------------
TOTAL                                 3546      0   100%
```

Figure 3.4: Example coverage report for the Ciw library.

of example cases are tested. Property based testing overcomes this problem by generating many inputs that the author of the tests are blind to. This forces the tests to check properties of the outputs, rather than the values themselves. This method leads testing closer to formal verification of the code.

A well formed property based test checks specific properties that captures everything of interest in the functionality of the code being tested. For example, property based tests for the function $g(t) = |t|$ would test the following properties:

- $g(t)$ is a number
- $g(t) \geq 0$
- $g(t) = t$ or $g(t) = -t$

The `hypothesis` library [101] is a Python library that works with the `unittest` library to perform property based testing, and is used in the Ciw testing suite. Its aim is to aid in finding failing edge-cases by generating numerous random example inputs to the unit tests.

Figure 3.5 shows the property based versions of the tests shown in Figure 3.3. The first a test ensures that errors are raised if the Python type 'None' is passed to the function, then all other tests correspond to those in Figure 3.3, however general random floats are given instead of concrete numbers.

3.2.2.4 Continuous integration

Continuous integration (CI) is a practice whereby developers push code to a shared repository regularly, and each time the code is pushed it is then built and tested in various environments. This process is usually done on an external CI server. The benefits of this practise are identifying bugs quickly, reducing problems when merging in contributions from collaborators, and adding transparency to the development process. In [33], there is a suggestion that these CI servers act as a way to integrate scientific works and algorithms from many researchers together and

```python
import unittest
from hypothesis import given
from hypothesis.strategies import floats

class TestExample(unittest.TestCase):
    @given(a=floats(), b=floats())
    def test_addition_hypothesis(self):
        if a is None or b is None:
            self.assertRaises(TypeError, add(a, b))
        self.assertEquals(add(a, b), a + b)
        self.assertTrue(add(a, b), add(b, a))
        self.assertTrue(isinstance(float, add(a, b)))
        self.assertRaises(TypeError, add('a number', a))
        self.assertRaises(TypeError, add('a number', b))
```

Figure 3.5: Hypothesis tests of the code tested using unit tests shown in Figure 3.3.

produce reliable and reproducible benchmarks for performance.

Travis CI [170] is one such service that integrates with GitHub, so that every push to GitHub is built and tested on Travis CI. Any pull requests are also tested on Travis, therefore no contributions need to be merged that will cause tests to fail.

An extension to this practice involves testing coverage with continuous integration. The Coveralls service [95] allows continuous integration servers such as Travis CI to report on the coverage of code, as well as the increase or decrease in coverage of each push.

Both Travis CI and Coveralls are used as part of the continuous integration set-up of Ciw.

3.2.3 Documentation

Writing and publishing documentation for software helps both users and developers with its understanding and increases ease of use. Creating and maintaining up to date documentation is important for software developers [96], and reading and using documentation forms an important part of learning and working with new software. Docstrings within the code aid with the code's readability and helps with navigation.

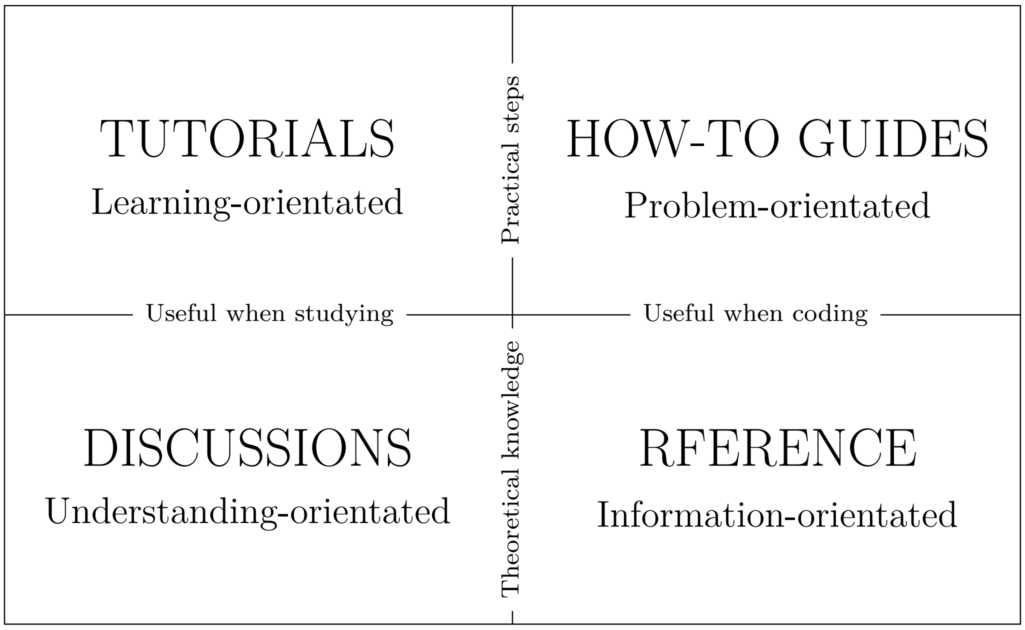


Figure 3.6: The Four-Quadrant structure of documentation given in [137].

These comments briefly explain the use of every class, method and function in the code.

User documentation needs to be more visible. Ciw uses Sphinx [135] to convert easily editable reStructuredText files into easily navigable and easily readable html, and also a neatly structured PDF file. This html is then hosted on ReadTheDocs [141], and made publicly available and is regularly updated. Sphinx and ReadTheDocs support LOCALE files, which offer the opportunity of publishing documentation in a variety of languages.

Ciw's documentation is available at `http://ciw.readthedocs.io/`. It is currently written in both English and Welsh. It is structured based on the recommended four-quadrant structure of [137]. These quadrants are Tutorials, How-to Guides, Reference, and Discussion, shown in Figure 3.6. Some seem to overlap, for example both Tutorials and How-to Guides give practical steps, however the nature of each quadrant is distinct.

The tutorials are lessons that assume no prior knowledge, and focus on results. The less distractions, explanations and discussions the better. How-to guides are goal orientated, giving step by step instructions while assuming the basics. More explanation can be given in these sections, however the aim is in solving real world problems. The guides are goal based and should be distraction free. References are technical descriptions of the machinery. There should be no explaining or showing, just describing. Finally discussion pages clarify or illuminate a particular topic. They give information setting the software in context, giving interested users a glimpse into the field itself.

Figure 3.6 also elucidates on the uses of the four quadrants. Tutorials and Discussion pages are useful when learning to use the software, whereas How-to guides and Reference pages are useful at the time of coding. Tutorials and How-to guides give practical knowledge, whereas Discussion and Reference pages provide theoretical understanding of the software.

The Ciw documentation is structured as follows, where the quadrants are named Tutorials, Guides, Reference, and Background:

- **Tutorials:** These are a series of tutorials aimed at users with no prior knowledge of the software. They walk the user through using the software without giving reasons nor explanations, with the aim of having achieved something by the end. In this case having defined and run a simulation of a given scenario, and found some key performance indicators.

- **Guides:** These are specific problem based guides showing how to implement the many features of Ciw, beyond the basics. Knowledge of the basics is assumed.

- **Reference:** These pages provide quick reference lists, for example the list of output results offered, or the list of distributions available.

- **Background:** These pages put the library in context, offer information on the mechanics of the code, and give some background on queueing theory.

Testing the documentation

Doctesting ensures that any code embedded into the code's documentation is up to date. This is not only important to ensure updated and accurate information, but also due to the common practice of users copying and pasting code from documentation directly for their own use. In the Ciw documentation, these tests are run using the `doctest` module in the Python standard library. It takes the file in which the documentation in written (for example .rst, .md, or as docstrings in a .py file), finds all instances of Python code written as if in a Python shell, and asserts their outcome with what was shown in the documentation.

3.3 Usage & features

Ciw's documentation has a number of examples and tutorials on its usage. This section will walk through the code required to simulate an $M/M/c$ system. Consider a queue with three servers, an arrival rate of $\lambda = 0.2$ and a service rate of $\mu = 0.1$. In order to simulate this system for 1440 time units, the code shown in Figure 3.7 can be typed into a Python shell.

```
>>> import ciw
>>> N = ciw.create_network(
...     Arrival_distributions=[['Exponential', 0.2]],
...     Service_distributions=[['Exponential', 0.1]],
...     Number_of_servers=[3])
>>> Q = ciw.Simulation(N)
>>> Q.simulate_until_max_time(1440)
>>> recs = Q.get_all_records()
```

Figure 3.7: Code to simulate an $M/M/c$ queue with Ciw.

First an instance of the generalised Network class is created defining all parameters of the system. This is a reusable object that contains all the information about the system to be simulated. This is then fed into a one-use Simulation object, which provides the engine behind the simulation mechanics. The `simulate_until_max_time()`

method takes in the maximum simulation time and runs the simulation from date 0 until that simulated date.

The `get_all_records()` method returns a list of records, one record for every service completed by each individual. This provides a comprehensive data set that allows flexibility in its analysis. Each record is a named-tuple, an object containing the following information:

- Customer I.D number
- Customer class
- Node at which service took place
- Arrival date to that node
- Waiting time
- Service start date
- Duration of service
- Service end date
- Length of time blocked
- Exit date
- Destination node (-1 if exiting system)
- Queue size at arrival
- Queue size at departure

As mentioned in the introduction, Ciw's main function is to handle networks of queueing systems with multiple nodes connected by transition probabilities; multiple classes of customers; and nodes with finite queueing capacities which display Type-I blocking.

In addition to these main properties, a number of other features are implemented in Ciw. This is important to support the bespoke nature of computer simulation, aiming to reflect reality as close as possible. These features, although originally required for specific projects, have been built in a general way such that they can find use in other projects with similar characteristics.

3.3.1 Scheduling disciplines

Scheduling disciplines are defined as the way in which customers are taken from the queue into service. That is the order in which customers in the queue are served. A number of disciplines are described in [157], including SIRO (service in random order) and LCFS (last come first serve).

Ciw allows three different scheduling disciplines.

- **FIFO**: First in first out. The server selects the customer at the head of the queue, the one that has waited longest. If a customer joins the queue before another customer, then they will also receive service before the other customer. This is the default scheduling discipline for Ciw.
- **IS**: Infinite server. All customers can be served immediately thus there is no need to queue. This is used if the users specifies an infinite number of servers.
- **PRIO**: Priority scheduling. The customer with the highest priority is chosen. In the case of ties, FIFO is used. Ciw allows each customer class to be given a priority level.

3.3.2 Variety of service & inter-arrival time distributions

Ciw allows users to choose from a variety of distributions that can be used to define service and inter-arrival time distributions. Currently there are seven continuous distributions that can be chosen, along with the deterministic distribution that returns a constant. These distributions are summarised in Table 3.1.

In addition to these standard distributions there are a number of other ways to define a distribution for service and inter-arrival times:

- Empirical distributions
- Custom discrete probability density functions

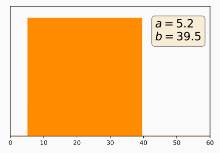
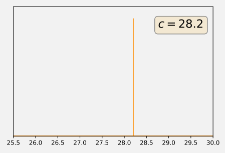
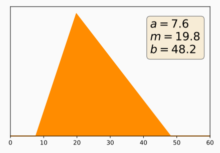
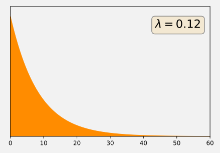
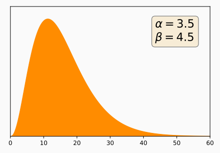
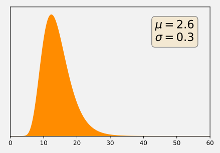
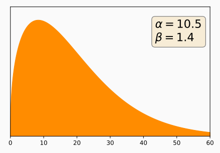
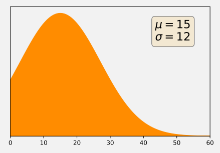
Distribution	***Parameters***	***Example***
Uniform	a = lower limit b = upper limit	
Deterministic	c = constant	
Triangular	a = lower limit m = median b = upper limit	
Exponential	λ = rate parameter	
Gamma	α = shape parameter β = scale parameter	
Lognormal	μ = mean of the normal σ = standard deviation of the normal	
Weibull	α = scale parameter β = shape parameter	
Truncated Normal	μ = mean of the normal σ = standard deviation of the normal	

Table 3.1: Table summarising the continuous distributions available in Ciw.

- Sequential distributions
- User defined distributions
- Time dependent distributions

An empirical distribution takes a set of values, either in the form of a Python list or a .csv file, and samples from the values given. This can be useful when trying to emulate real-life data, especially when the data doesn't seem to follow one of the standard continuous distributions listed in Table 3.1. Custom discrete probability distributions can also be very useful, they sample from a set of values with assigned probabilities. Sequential distributions take a list of values and iteratively returns the next observation in the list in sequence. A sequential distribution is cyclic, and thus will return to the beginning of the list once it has run out of values. These are very convenient for use in the testing framework. Both user defined distributions and time dependent distributions allow the user to write their own Python function that will return observations, either stochastically or deterministically. Time dependent functions can also take in a time parameter, that uses the simulation's current time to determine the next observation.

3.3.3 Dynamic customer classes

This feature enables customers to stochastically change their customer class upon completion of service. This enables more complex behaviour to occur in the model, and can be utilised in a manner that can overcome the memoryless property of queueing networks.

As an example of this, consider a one node network in which customers must rejoin the queue for further service once. However the first time a customer is served at that node they have a different service distribution to the second time.

This could be difficult to model as Ciw is not a process-based simulation and so does not keep track of each individual's history. The event scheduling approach is memoryless. However this can be modelled by considering the system with three customer classes: Class 0 are the customers who are entering the node for the first time; Class 1 customers are those who are entering the node for the second time; and Class 2 customers are those who leave the system.

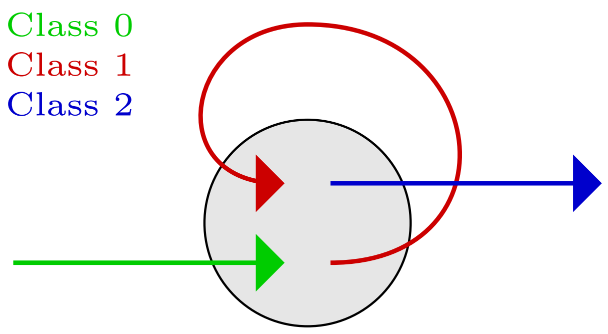


Figure 3.8: Illustration of an individual's behaviour when exposed to dynamic customer classes.

Consider a matrix M where the rows and columns correspond to customer classes. If a customer of class r finishes a service, then there is a probability $M_{(r,c)}$ of that customer changing into a customer of class c. The following matrix is defined:

$$M = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

According to this matrix Class 0 customers have probability 1 of becoming Class 1 customers after service. Similarly Class 1 and Class 2 customers have probability 1 of becoming Class 2 customers after service.

In this example, consider that customers of Class 1 are certain to rejoin the node; Class 2 customers will never rejoin the node; both Class 1 and 2 customers will never arrive at the node externally; and Classes 0 and 1 have different service distributions. Then implementing the class change matrix M is sufficient to achieve the required behaviour, overcoming the inherently memoryless property of queueing networks. An individual customer's behaviour at this node is illustrated in Figure 3.8.

3.3.4 Server schedules

This feature allows for nodes with variable numbers of servers. That is, the servers work to a schedule where during certain dates, or shifts, there are different amounts of servers on duty. The server schedules are cyclic, and so the shifts repeat each time the shift has undergone a cycle. An example server schedule is shown in Table 3.2.

Shift	0 – 40	40 – 65	65 – 110	110 – 150
Number of Servers	1	0	2	3

Table 3.2: An example server schedule.

This schedule indicates that between the dates 0 and 40 there should be 1 server active at that node; no servers should be active between dates 40 and 65 then there should be 2 active servers between 65 and 110; and 3 active servers between 110 and 150. As the schedules are cyclic the schedule repeats, and so between dates 150 and 190 there will be 1 server on duty, between 190 and 215 there won't be any servers on duty, and so on until the simulation ends. In this case the cycle length is 150 time units.

It is important to note that at the shift boundaries a full shift swap over happens. That is every server from the previous shift goes off duty, and then a new set of servers begins their shift. Ciw has however implemented two disciplines for the server schedule boundaries, pre-emptive schedules and non-pre-emptive schedules:

- During pre-emptive schedules, at the time of a shift change all customers currently in the middle of service are abandoned, and their service will be picked up again by a new server. This requires that their service time is re-sampled and their service time begins again from the start. If more customers have their service abandoned than there are new servers, then they will have to wait again, although they take priority over those customers who have not began service yet.
- During non-pre-emptive schedules, at the time of a shift change all servers currently in the middle of service must remain on duty until that service ends. This may cause an overlap of servers at the beginning of the shift.

An example of how the overlaps in the schedule in Table 3.2 may look, using the non-pre-emptive discipline is shown in Figure 3.9. Ciw also records all overtime done by servers caused by this phenomenon.

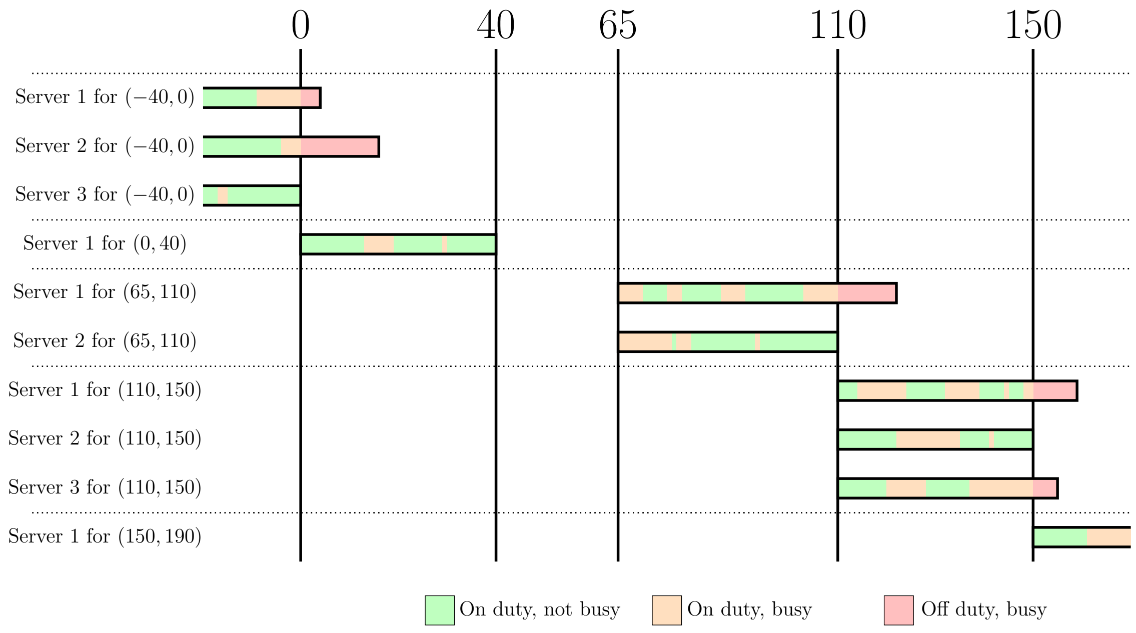


Figure 3.9: Example of the overlaps in shifts that may occur in the schedule shown in Table 3.2 using the non-pre-emptive discipline.

Thus, whichever discipline is chosen the two schedules shown in Table 3.3 are not equivalent. Under the pre-emptive discipline the second schedule would cause customers who are in the middle of their service at time 10 to have their service time re-sampled and restarted; and under the non-pre-emptive discipline the second schedule may have an overlap of servers directly after the change of shift that occurs at time 10.

Shift	$0-20$	$20-30$
Number of Servers	1	2

Shift	$0-10$	$10-20$	$20-30$
Number of Servers	1	1	2

Table 3.3: Comparing two seemingly equivalent schedules.

3.3.5 Batch arrivals

In general it is assumed that customers arrive to a node alone. However in some cases it is more appropriate to model customers arriving in groups, or batches. Ciw allows batch arrivals, and in particular allows probabilistic batch sizes. That is it allows users to define a discrete distribution to sample the batch size from, for each node of the network, and each customer class.

A similar effect can be achieved by sampling inter-arrival times of zero, however this is impossible if using continuous distributions. This feature allows for greater control of the batch arrival process. Note however that the presence of batch arrivals increases the prevalence of simultaneous events, discussed further in Section 4.3.

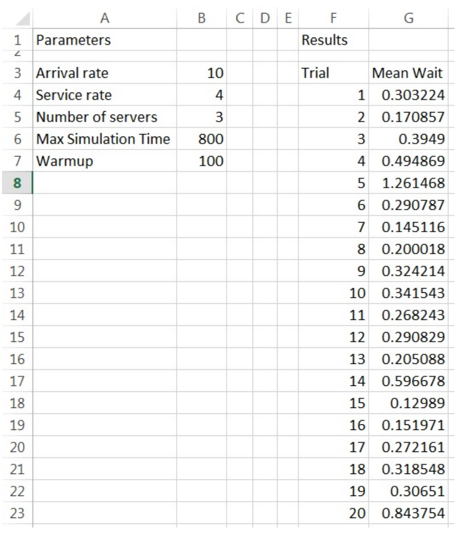
3.3.6 Baulking customers

Ciw allows for customers to baulk [5, 6], that is choose not to join the queue after arrival because the queue is deemed to be too long. This decision to baulk is usually modelled as being stochastic, to account for different customers' preferences. This is usually defined by a baulking function $b(m)$, that describes the probability of arriving customers baulking given that there are m customers already at the node.

Ciw allows flexibility and control in this by allowing users to define separate baulking functions for each node and for each customer class. Furthermore, due to Ciw's position in a Python ecosystem, $b(m)$ is defined as a Python function, allowing bespoke baulking functions to be written, which can be as complex or as simple as required.

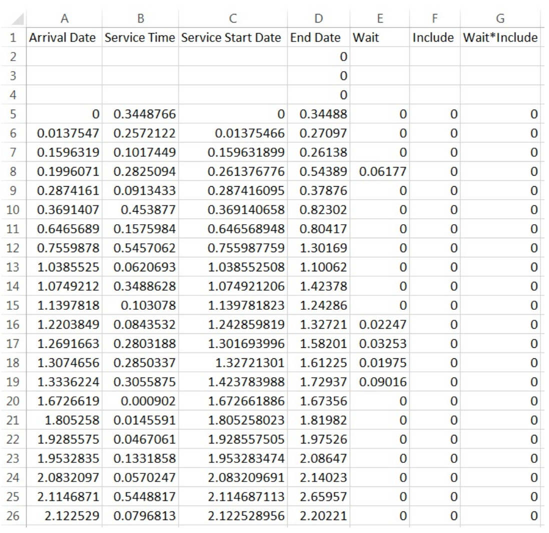
3.4 Comparisons with other simulation tools

Ciw is compared to other simulation tools, with comparisons taken on their appropriateness for conducting reproducible research and run times. Five other popular simulation frameworks from across the spectrum corresponding to user interface (that shown in Figure 3.1) were chosen for comparison. The frameworks chosen were:



	A	B	C	D	E	F	G
1	Parameters					Results	
2							
3	Arrival rate	10				Trial	Mean Wait
4	Service rate	4				1	0.303224
5	Number of servers	3				2	0.170857
6	Max Simulation Time	800				3	0.3949
7	Warmup	100				4	0.494869
8						5	1.261468
9						6	0.290787
10						7	0.145116
11						8	0.200018
12						9	0.324214
13						10	0.341543
14						11	0.268243
15						12	0.290829
16						13	0.205088
17						14	0.596678
18						15	0.12989
19						16	0.151971
20						17	0.272161
21						18	0.318548
22						19	0.30651
23						20	0.843754

(a)



	A	B	C	D	E	F	G
1	Arrival Date	Service Time	Service Start Date	End Date	Wait	Include	Wait*Include
2				0			
3				0			
4				0			
5	0	0.3448766	0	0.34488	0	0	0
6	0.0137547	0.2572122	0.01375466	0.27097	0	0	0
7	0.1596319	0.1017449	0.159631899	0.26138	0	0	0
8	0.1996071	0.2825094	0.261376776	0.54389	0.06177	0	0
9	0.2874161	0.0913433	0.287416095	0.37876	0	0	0
10	0.3691407	0.453877	0.369140658	0.82302	0	0	0
11	0.6465689	0.1575984	0.646568948	0.80417	0	0	0
12	0.7559878	0.5457062	0.755987759	1.30169	0	0	0
13	1.0385525	0.0620693	1.038552508	1.10062	0	0	0
14	1.0749212	0.3488628	1.074921206	1.42378	0	0	0
15	1.1397818	0.103078	1.139781823	1.24286	0	0	0
16	1.2203849	0.0843532	1.242859819	1.32721	0.02247	0	0
17	1.2691663	0.2803188	1.301693996	1.58201	0.03253	0	0
18	1.3074656	0.2850337	1.32721301	1.61225	0.01975	0	0
19	1.3336224	0.3055875	1.423783988	1.72937	0.09016	0	0
20	1.6726619	0.000902	1.672661886	1.67356	0	0	0
21	1.805258	0.0145591	1.805258023	1.81982	0	0	0
22	1.9285575	0.0467061	1.928557505	1.97526	0	0	0
23	1.9532835	0.1331858	1.953283474	2.08647	0	0	0
24	2.0832097	0.0570247	2.083209691	2.14023	0	0	0
25	2.1146871	0.5448817	2.114687113	2.65957	0	0	0
26	2.122529	0.0796813	2.122528956	2.20221	0	0	0

(b)

Figure 3.10: Screenshots of the spreadsheet model developed in Microsoft Excel: 3.10a shows parameters and results, while 3.10b shows the mechanics of one trial.

- C++ (version 11, compiled using g++ 4.2.1)
- Spreadsheet modelling (implemented in Microsoft Excel 2013)
- SimPy (version 3.0.10, using Python 3.5.1)
- Ciw (version 1.1.3, using Python 3.5.1)
- AnyLogic (AnyLogic 8 University 8.1.0)
- SIMUL8 (SIMUL8 2014 [Exclusive EDUCATIONAL SITE Edition])

For comparison, a model of an $M/M/3$ queue, with arrival rate $\lambda = 10$, and service rate $\mu = 4$ was built in each framework. The models were run for 800 time units, with a warmup time of 100 time units. The aim was to find the average waiting time in the queue.

The C++, SimPy, and Ciw models can be found in Appendix A. The C++ model was written by Miss Asyl Hawa. Microsoft Excel was chosen as the spreadsheet software. Screenshots of the spreadsheet, SIMUL8, and AnyLogic models are shown in Figures 3.10, 3.11, and 3.12 respectively. All models are archived and can be found at [123]. Some initial observations are summarised in Table 3.4.

The capabilities of a spreadsheet model were not found to align with the expectations

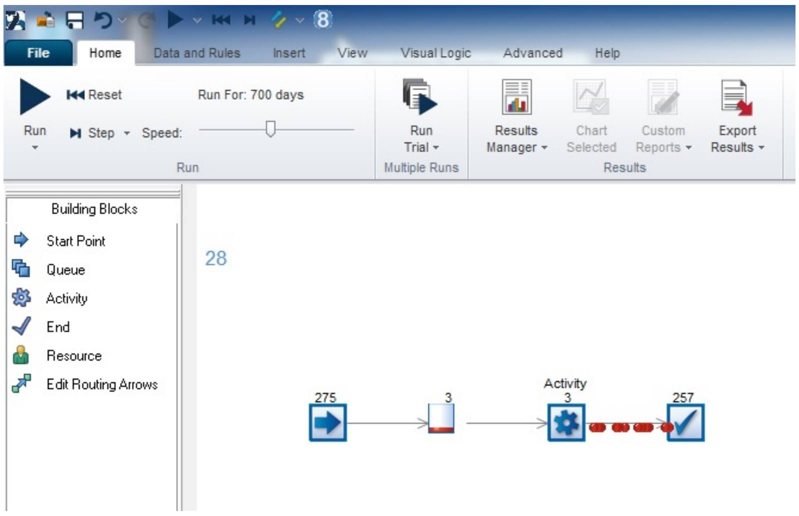


Figure 3.11: Screenshot of the SIMUL8 model.

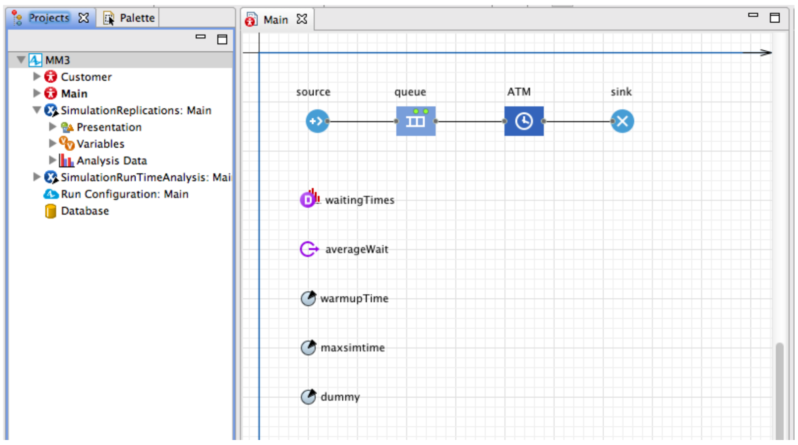


Figure 3.12: Screenshot of the AnyLogic model.

	Version controllable	Licence	Modifiable	GUI	Animation	Support
C++	✓	GNU GPL free licence.	Bespoke models	N/A	N/A	N/A
Spreadsheet modelling	✗	Depends on software.	Limitations to what can be modelled.	N/A	✗	N/A
SimPy	✓	MIT	Extensible & modifiable source code.	Limited GUI available for running models.	✗	Online documentation
Ciw	✓	MIT	Extensible & modifiable source code.	✗	✗	Online documentation
AnyLogic	With professional licence only.	Limited PLE version available, otherwise commercial.	Extend with Java.	✓	✓	Online documentation & paid training.
SIMUL8	✗	Commercial	Extend with Visual Logic.	✓	✓	Online documentation & paid training.

Table 3.4: Summary of the comparisons between six simulation frameworks.

of research best practice. Seeds cannot be set, thus reproducibility is impossible. The resulting model has very low interpretability, unless set out as a 'black box' model where parameters are input and results are output. However, this style of model hinders model understanding and communication. In fact the model, including input, output and mechanics, are all shown in Figure 3.10, and yet the model is still very difficult to interpret. This in turn leads to low confidence in results, a well reported phenomenon [189] with the use of spreadsheets. In addition, most data had to be handled manually, further impeding reproducibility.

Building a model with a GUI, such as with SIMUL8 and AnyLogic, may ease the model development process although care should be taken to follow best practice. The model may be more accessible given its visual nature which can aid with communication, although knowledge of the software is required to read much of the model parameters as many of these are hidden behind objects and menus (for example, Figures 3.11 and 3.12 do not in themselves show basic model parameters such as number of servers, arrival and service rates). The binary files which represent the SIMUL8 models and the restrictive commercial licences on both SIMUL8 and AnyLogic inhibit accessibility and model sharing, and thus reproducibility.

The bespoke model developed in C++, given its compiled nature and that the model was not held back by unused features, used the least (by far) computation time. The model, a script, is shareable and can be put under version control. Readability is hindered here as all details, including internal simulation mechanisms, are shown which may make communications with non-specialists challenging.

The models developed with SimPy and Ciw are version controllable and shareable, and thus ideal for replicable results. Compared to the model developed in C++, much of the simulation mechanics are hidden from the user and pre-tested. This aids with model communication, validation, and reduces error. I believe that the Ciw model is more readable than the SimPy version. Much of the simulation mechanics are on show with SimPy, which increases its flexibility but reduces readability,

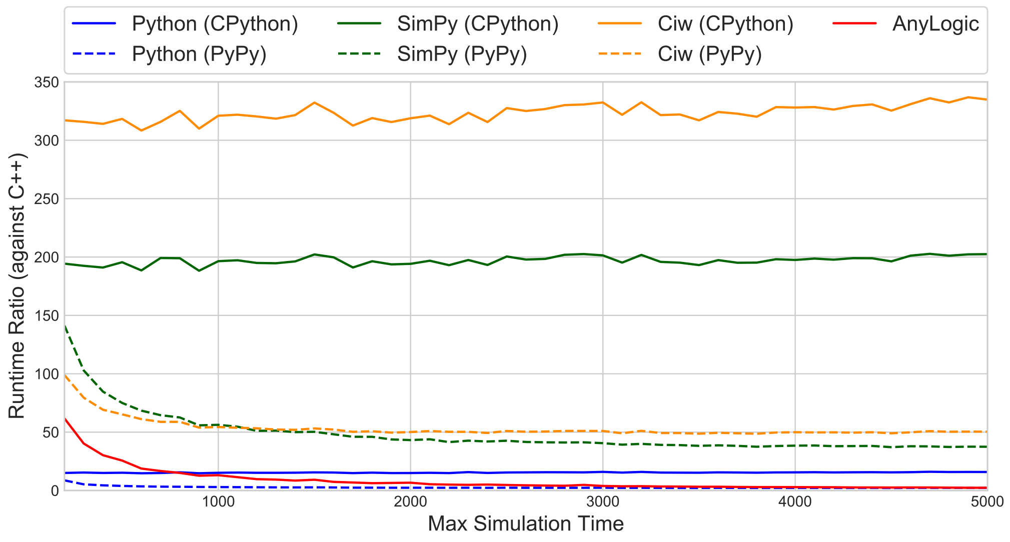


Figure 3.13: Comparison between the runtimes of the C++ model, bespoke Python model, the SimPy model, the Ciw model, and the AnyLogic model.

whereas Ciw prioritises readability.

Runtime analysis was carried out on the software for which it was possible: C++, SimPy, Ciw, and AnyLogic. All analyses were performed on an Apple MacBook Air with 1.4GHz Intel Core i5 processor, OS X 10.9.5 (13F34), with 4 GB 1600 MHz DDR3 memory. SIMUL8 could not be included in the analysis; due to licensing restrictions, the model could not be run on the same machine as the others (for consistency). Importantly, the version used does not include tools to record computational runtime. Figure 3.13 shows the average runtimes from five runs, as a ratio of the fastest running model C++, for maximum simulation times from 200 to 5000 time units. Also included in the analysis is a bespoke Python model (an equivalent model to the C++ model but coded in Python). The three Python models were run through two interpreters: the standard CPython interpreter (version 3.5.1), and PyPy [166] (version 5.1.1), which is an alternative implementation of Python with a Just In Time compiler that improves runtimes.

It is worth noting that the runtime recorded for the AnyLogic model does not take into consideration experiment initialisation time, nor does it include application launch and model loading. Therefore in practice, running AnyLogic models take longer than that which is reported in the plot.

Figure 3.13 reflects what has been discussed in much of the literature [92, 147] that bespoke models coded using programming languages yield faster running models than simulation packages. The simulation packages here, SimPy and Ciw, carry around many unused features that may slow down the model. Running Ciw with PyPy greatly improves performance and it seems that PyPy is mainly affected by the simulation initialisation time, as running the simulation for longer sees the performance approaching that of the C++ model. Furthermore, using Python's inbuilt parallel processing library, it is straightforward to parallelise trials of Ciw. Speed, however, is not a priority of Ciw, favouring instead code readability, and the library's ability to enable reproducible scientific research. In practice, the computational runtime for Ciw models does not hinder practical use in research, though may be disadvantageous in complex systems or other models where speed is a priority.

3.5 Summary

This chapter has outlined the need for and the development of an open source discrete event simulation library in Python, Ciw. Issues with commercial off-the-shelf simulation software with GUIs are discussed, these include their lack of transparency, restrictive licences, and shortcomings in facilitating reproducible research. An alternative approach is proposed, open source software with script based models, that is deigned with reproducibility in mind. This allows for readable, reusable, extensible, modifiable, and testable models that can be put under version control. Furthermore, the development process of the software itself follows best practice, with full documentation, test coverage and modularity.

Some of Ciw's modelling features are discussed including service discipine, dynamic customer classes, batch arrivals, baulking customers and server schedules. Finally there is a comparison between Ciw and other simulation software options from a spectrum of user interface. Ciw, Simpy, SIMUL8, AnyLogic, spreadsheet modelling in Excel, and custom scripts in C++ and Python are compared in terms of both performance and their aptness of reproducible research. Packages using scripted models, Ciw and Simpy, perform the worst in comparison to both custom scripts and commercial solutions; however they do prove most apt for enabling and encouraging reproducible simulation models.

Chapter 4

The Software Mechanics

This chapter will discuss some of the source code and simulation mechanics that allow Ciw to work.

This chapter is structured as follows:

- Section 4.1 outlines the event scheduling approach used in Ciw, and contrasts it to the similar three phase approach.
- Section 4.2 details how this approach is implemented in Ciw.
- Section 4.3 describes a potentially problematic occurance in discrete event simulations whereby two events are scheduled to occur simultanously, and describes Ciw's method of dealing with them.
- Section 4.4 describes potential biases with scheduling events simultaneously under floating point arithmetic, and gives Ciw's solution to this problem.
- Section 4.5 discusses the implementation of the deadlock detection method described in Chapter 2.
- Section 4.6 summarises the contents of this chapter.

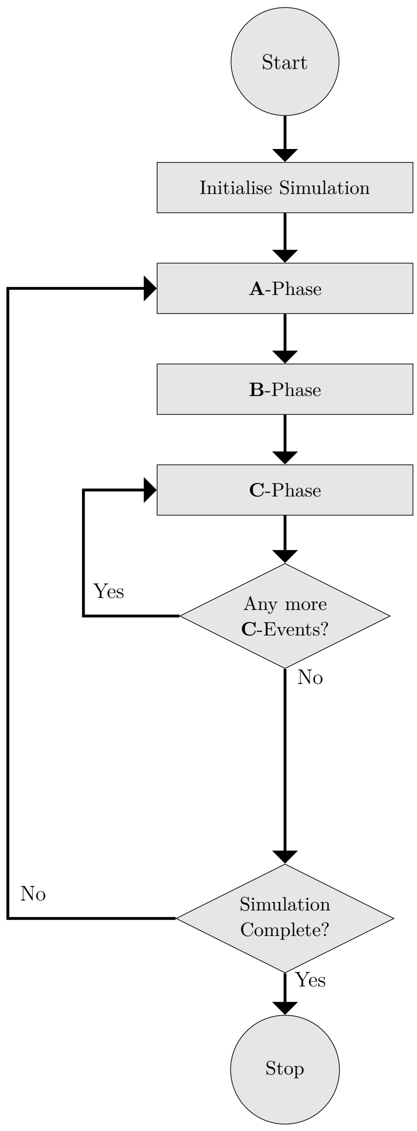
4.1 Event scheduling simulation approach

Ciw implements the event scheduling simulation approach. Event scheduling is a variant of the standard three-phase approach, widely discussed in the literature. It contrasts with another popular approach, process orientated simulation.

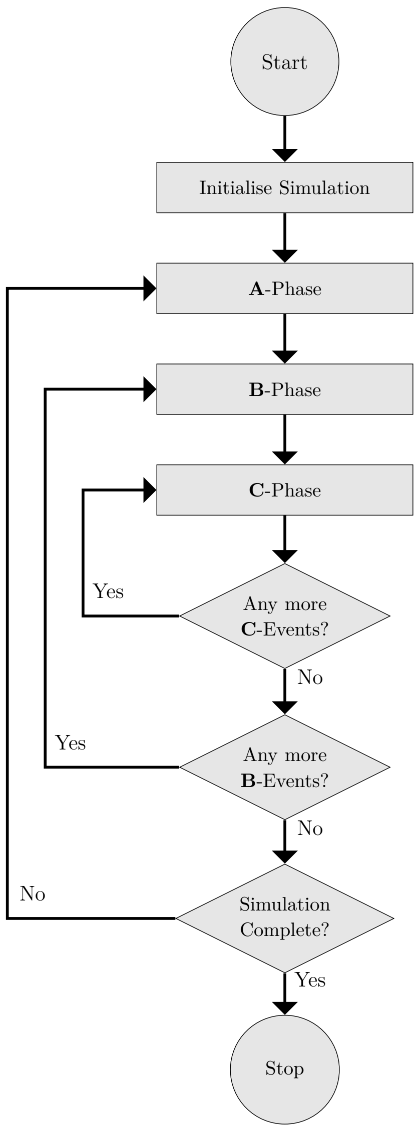
A comparison of these approaches is given in [134]. In process orientated simulation, entities have their complete journey through the simulation determined at the beginning. That is there is a complete chronological sequence of operations for each entity to undertake during their time in the system. These events may be temporarily or conditionally delayed, for example while waiting for a server to become free, however each entity follows its own process from the start to completion of the simulation.

The three-phase simulation approach is described in [147]. Here there is no attempt to create a sequence of events for entities to follow. The focus here is on the activities and events themselves, independent of one another, rather than the entities. The flow of the three-phase approach is shown in Figure 4.1a:

- In the **A**-phase the simulation clock is moved to the time of the next event.
- In the **B**-phase *all* scheduled events for that time are carried out.
- In the **C**-phase we check whether some of the events just carried out have caused any more conditional events to occur at the present time, and *all* of these are carried out.
- At the decision node 'Any more **C**?', we check whether the **C**-events that just occurred has caused any more **C** events to need to occur.
- Once there are no more **C**-events to be carried out, we check whether the simulation has completed. If the simulation has obtained some property (for example the next time an event should occur is after the scheduled end of the simulation run), then the simulation ends. Else we go back to the **A**-phase



(a) Flow diagram of the three-phase simulation approach.



(b) Flow diagram of the event scheduling simulation approach.

Figure 4.1: Differences between the three-phase and event scheduling simulation approaches [147].

and move the clock along to the time of the next scheduled event.

Note that both **B**-events and **C**-events can trigger other **C**-events to happen, sometimes resulting in a sequence of events that all happen at the same date. In the same way, both **B**-events and **C**-events can trigger other **B**-events to be scheduled for the future.

In the event scheduling approach, this procedure is modified slightly, shown in Figure 4.1b:

- In the **A**-phase the simulation clock is moved to the time of the next event.
- In the **B**-phase *one* of the scheduled events for that time is chosen to be carried out.
- In the **C**-phase we check whether the event just carried out has caused any more conditional events to occur at the present time, and *one* of these is chosen to be carried out.
- At the decision node 'Any more **C**?', we check whether the **C**-events that just occurred has caused any more **C** events to need to occur, or if there are any original **C**-events scheduled for the present time that need to be carried out.
- At the decision node 'Any more **B**?', we check whether there are any more **B**-events scheduled at the present time that need to be carried out.
- Once there are no more **B** or **C**-events to be carried out, we check whether the simulation has completed. If the simulation has obtained some terminating property, then the simulation ends. Else we go back to the **A**-phase and move the clock along to the time of the next scheduled event.

This slight variation on the three-phase approach does not affect the simulation time that events are carried out. It does however change the order of simultaneously scheduled events to a more logical sequence. The event scheduling approach does leave some ambiguity about the order of some simultaneous events however, and this is discussed further in Section 4.3.

All events in Ciw are categorised into **B** and **C**-events. **B**-events are those that are scheduled ahead of time, while **C**-events are those that were previously not scheduled, but must occur immediately due to a **B** or **C**-event having occurred. Table 4.1 below shows some of the possible conditional **C**-events that arise from other events. Note that this list isn't exhaustive, for example a baulking customer may leave the system after arriving, and shift changes are **B**-events that can cause many conditional events to be triggered. As more features are added the chain of possible events may also grow.

4.2 Implementation

In order to implement the event-scheduling approach described above Ciw makes full use of Python's object-orientated nature. The code is structured around classes and methods that manipulate objects. Ciw's hierarchy of classes and how they interact is shown in Figure 4.2.

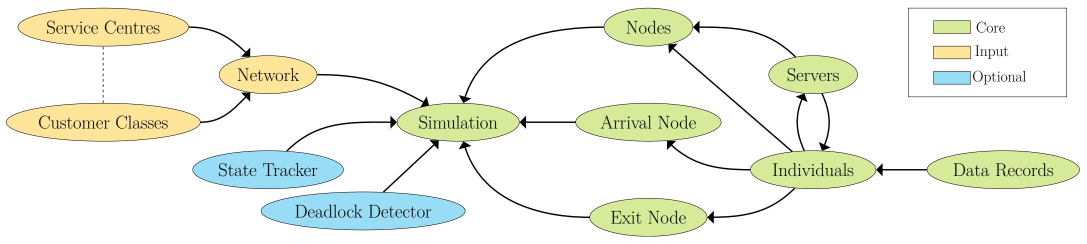


Figure 4.2: The structure of Ciw's code.

A simulation engine is entirely contained within a `Simulation` object, which contains all other objects that make up the simulation, and also runs the main simulation loop method, and a method to collect all data records. The entities that move within the system are `Individual` objects. These entities move from node to node, wait for and spend time in service. They also carry a list of `DataRecord` objects. Each `DataRecord` contains information about one particular service of one particular individual.

Event	***Conditional Event***	***Cause***
External arrival	Nothing	The customer arrives and needs to queue before starting service.
External arrival	Start service	The customer arrives and can begin service immediately as there is no queue.
Finish service	Release/Move customer	A customer finishes service, there is sufficient queueing space at the next node, and so moves.
Finish service	Nothing	A customer finishes service, but there is insufficient queueing space at the next node, and so the customer becomes blocked.
Release/Move customer	Start service	A customer moves to the next node and can begin service immediately as there is no queue.
Release/Move customer	Start service	A customer moves from the current node, allowing a customer at the front of the queue to start service.
Release/Move customer	Release/Move customer	A customer moves from the current node, allowing enough queueing space from a blocked customer to now enter that node.
Release/Move customer	Nothing	A customer moves to the next node where there is a queue, and so service does not start immediately. There is no one in the current node's queue to start service.
Release/Move customer	Nothing	A customer exits the system, and there is no queue at the current node and so no-one starts service.

Table 4.1: Table listing some possible conditional event triggers.

Service centres and their queues correspond to `Node` objects. These important objects contain most of the methods that together carry out the **B**-events and **C**-events. They contain all the individuals who are either in service or waiting for service at that service centre, and also contain a number of `Server` objects. Two additional ‘dummy’ nodes are connected to the network, an `ArrivalNode` and an `ExitNode`. The `ArrivalNode` object spawns new individuals when an external arrival occurs. The `ExitNode` object collects all individuals who leave the system.

The `Server` objects attach and detach from `Individual` objects whenever that individual begins service with that server or is released by the server. Whenever an individual becomes blocked, the server remains attached to that individual until the condition for release is satisfied. A `Server` may only be attached to one `Individual` at a time, and similarly an `Individual` may only be attached to one `Server` at a time.

Two optional objects are also included in the `Simulation`: a deadlock detector object, discussed in Section 4.5, and a various options for a state tracker object, that track the state of the system as the simulation proceeds.

The input object `Network` contains all the parameters of the system, and this tells the `Simulation` object how to construct and arrange the nodes of the system. This object is made up of `ServiceCentre` and `CustomerClass` objects, that contain information and parameters about the system. The information contained in these objects are structurally connected, and so a variety of helper functions are provided to construct a coherent `Network` object.

Figures 4.3, 4.4 and 4.5 show flow diagrams of the processes that drive the simulation in Ciw. Together they describe the mechanisms of the simulation code, although the source code itself gives the exact details (found at GitHub and Zenodo). The diagrams are colour-coded to show which objects are undertaking each task: $\boxed{\text{green ♥}}$ actions indicate that the `Simulation` object carries out that task, $\boxed{\text{orange ♠}}$ actions indicate that the `ArrivalNode` carries out that task, $\boxed{\text{blue ♣}}$ actions indicate that the current `Node` carries out that task, and $\boxed{\text{purple ♦}}$ actions indicate that the destination `Node` carries out the task. $\boxed{\text{White □}}$ actions indicate that a choice of objects, or multiple objects, carry out the task.

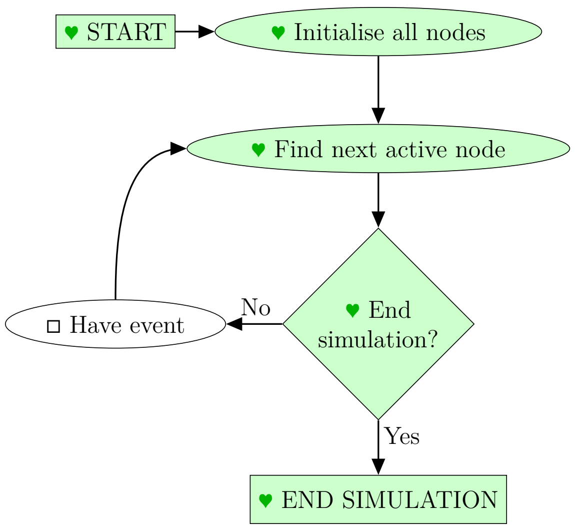


Figure 4.3: Flow diagram of Ciw's main simulation loop.

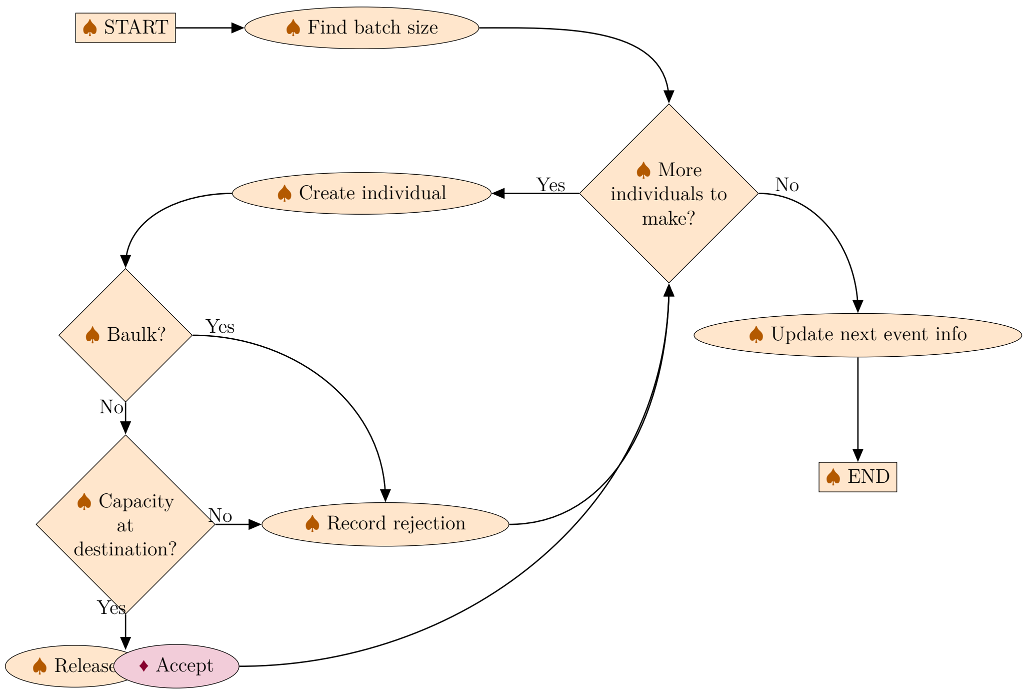


Figure 4.4: Flow diagram of Ciw's `ArrivalNode` having an event.

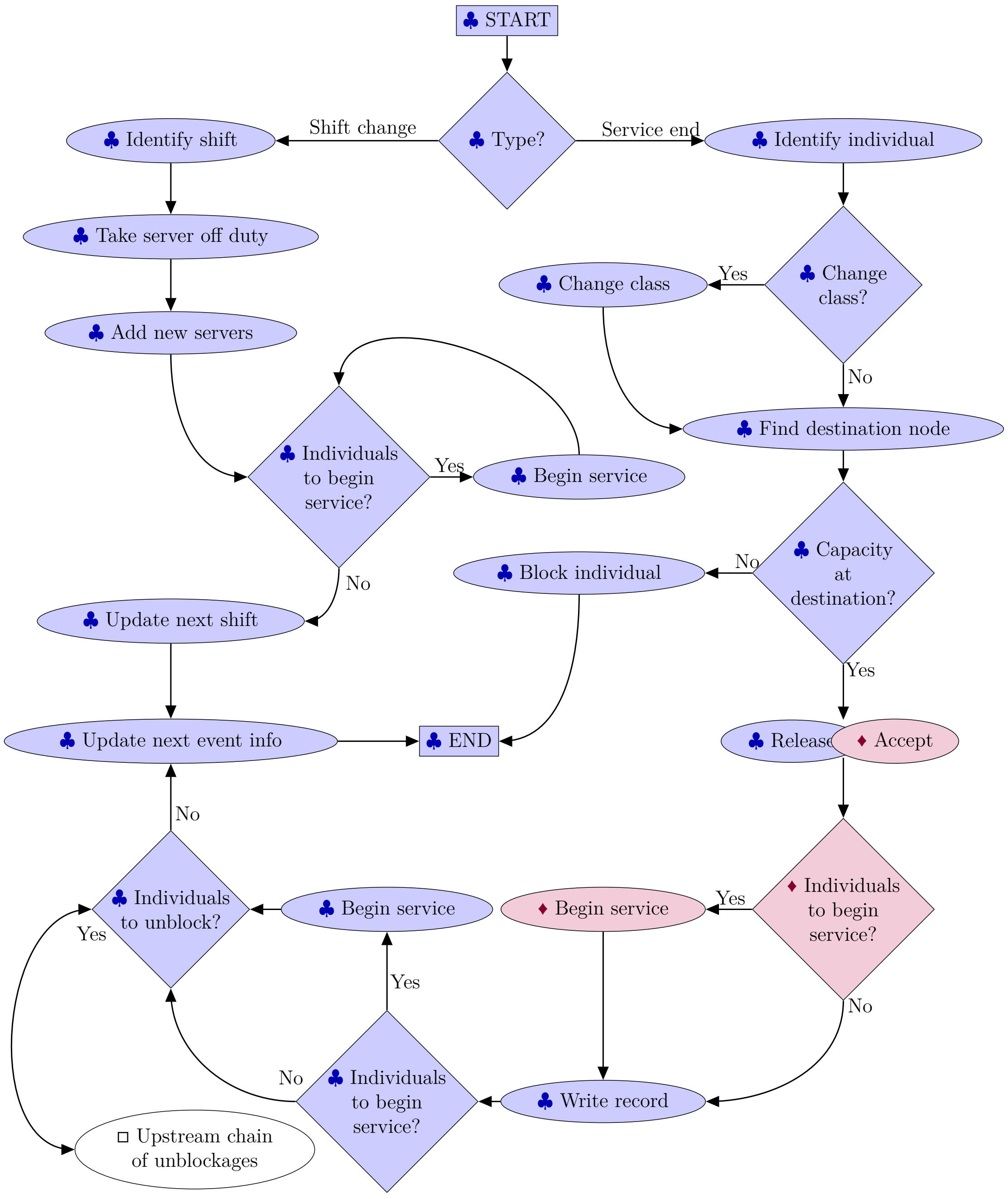


Figure 4.5: Flow diagram of Ciw's general `Node` having an event.

The main simulation loop, shown in Figure 4.3, consists of repeatedly selecting nodes and having events until the simulation ending condition is satisfied. These 'events' correspond to the **B**-events of the event scheduling approach, of which three can occur, described below. Note that conditional **C**-events are not coded explicitly, but occur as consequences of **B**-events being carried out, shown in the flow diagrams.

- An external arrival. This event is carried out by the `ArrivalNode`, and its process is shown in Figure 4.4. Here a batch size is sampled, and for each individual in that batch, it is checked if they baulk or are rejected for service due to capacity constraints. If they qualify for service, the `ArrivalNode` then releases that individual, while their destination `Node` accepts that individual.

- An individual finishes service. This event is carried out by the `Node` in which that customer finished service, shown in Figure 4.5. That individual's customer class may change, and may be sent to another node if that destination node's capacity allows it. This requires the current node to release the individual, and the destination node to accept that individual, beginning service if possible. The current node now needs to check if there are any waiting customers who can now begin service, and if there are any blocked customers that can now be unblocked due to the recent departure. This may cause an upstream chain of unblockages through the network.

- A server shift change occurs. This event is carried out by the `Node` object, shown in Figure 4.5. Servers are taken off-duty (and 'killed' if empty or if pre-emptive schedules are used; otherwise they will wait until the end of their current service to be 'killed') and new servers are added. This may cause waiting individuals to be able to be served, so they begin their service. Then the server schedule is updated.

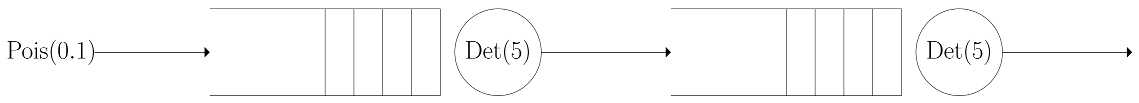


Figure 4.6: Example of a network where simultaneous **B**-event may occur.

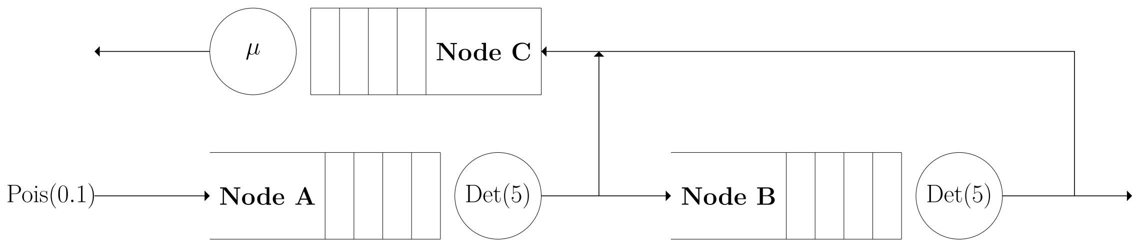


Figure 4.7: Example network where the order of simultaneous events matters.

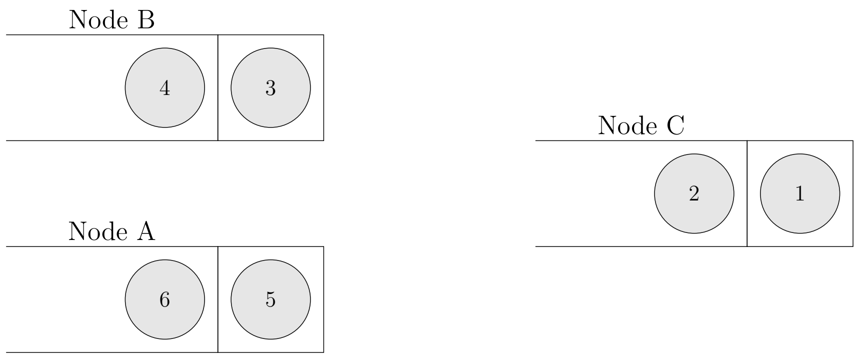
4.3 Simultaneous events

In general the processes shown in Figures 4.3, 4.4 and 4.5 ensure that event are carried out in the correct order according to the event scheduling approach. When using discrete or deterministic service time distributions, some systems may have a non-zero probability of having two scheduled **B**-events simultaneously. An example of how this situation might occur is shown in Figure 4.6.

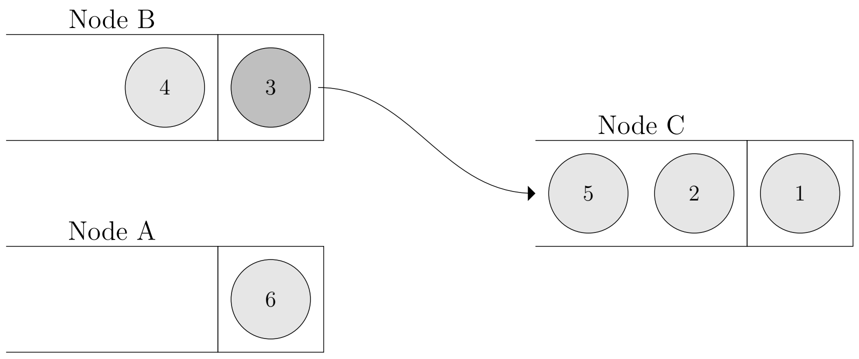
Consider the case when the second node is empty and there is at least two customers at the first node. Service times at both nodes are deterministic lasting 5 time units. At time t the customer in service at the first node finishes service and joins the second node, immediately beginning service. A customer waiting at the first node can now begin service at that node. At time $t + 5$ the customers in service at both the first and seconds nodes finish service independently, causing two **B**-events to occur simultaneously.

Ciw must carry both these out separately, without moving the clock forward. The order in which these are carried out does matter however. Consider the system illustrated in Figure 4.7.

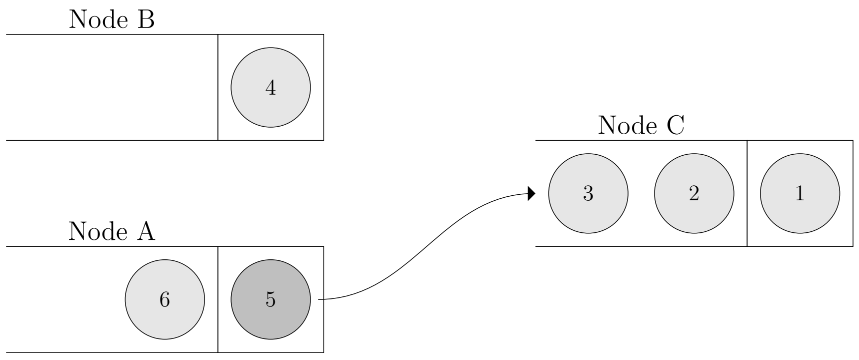
Consider the case when the node C has space for only one more customer to queue. Due to the system described in Figure 4.6, the two other nodes have customers finishing service simultaneously. Now both these customers wish to transition to the top node. The order in which these events take place can determine which customer get blocked and which customer doesn't. This is illustrated in Figure 4.8.



(a) Node A and B finish service simultaneously.



(b) Event at node A occurs first: customer 3 is blocked.



(c) Event at node B occurs first: customer 5 is blocked.

Figure 4.8: Two cases where the order of simultaneous events affects the outcome.

These types of simultaneous events are well known, and is in fact a feature of the three-phase and event scheduling simulation approaches [147]. In [139] problems arising from simultaneous events are used to illustrate that discrete simulations are in fact attempting to model a continuous system; where events that last a non-negligible amount of time are modelled as instantaneous. Both a theoretical and an real world example of how simultaneous events within simulation models can vastly effect outcomes are discussed in [185].

In general simulation protocols put all events to be scheduled at the current date into a 'current events list' [151]. The sequencing of this list is generally determined by some form of event priorities with ties broken by FIFO, that is carried out in the order that the simulation protocol scheduled them [147, 151]. Priorities may be due to event type, or how the event was scheduled. The authors of [74] categorise these sequencing or tie-breaking mechanisms as either 'user-consistent' or 'arbitrary', and as either 'deterministic' or 'nondeterministic'. In arbitrary mechanisms the sequence is determined internally by the simulation protocol without control from the user, whereas with user-consistent mechanisms the modeller has some form of control or knowledge over the sequencing technique. Deterministic tie-breaking mechanisms ensure that every execution of the mechanism results in identical results, ensuring reproducibility, whereas non-deterministic mechanisms result in stochastic outcomes.

An explicit tie-breaking or sequencing mechanism may not produce a satisfactory solution however. In [139] we are reminded that these event selection algorithms are an artificial element and dependant upon the simulation protocol itself. They have no corresponding meaning in the real system being modelled. This is further asserted in [185], where the 'Event Simultaneity Assertion' is introduced. This states that if there are N simultaneous events, then the true outcome is the average outcome over the $N!$ orderings of these events. Finally in [133] methods for investigating all sequences of simultaneous events are discussed. Here a branching process is introduced, where given a tie, the simulation branches, and many clones of the simulation are run in which every possible combination of simultaneous events are explored.

Due to the code structure of Ciw, two types of simultaneous events exist: simultaneous events within nodes, and simultaneous events between nodes. Nodes in this case refer to the nodes within a queueing network, in addition to a dummy arrival node which spawns new customers. Simultaneous events within nodes means that two or more events are scheduled to take place at the same date at the same node. Simultaneous events between nodes mean that two or more nodes have an events scheduled to take place at the same date. This distinction is important for Ciw, as the code first chooses a node to have an event, and then chooses which event within that node is to occur.

Another variant of this problem can cause some customers to wait while others immediately begin service. Consider the same network as in Figure 4.7, now with Node C empty. Again, due to the situation described in Figure 4.6, nodes A and B release customers simultaneously, and both transition to Node C. As there is only one server at Node C, the customer released first will immediately begin service, while the other will have to wait in the queue.

Yet another variant involves one node having simultaneous events. Consider the example above but where Node C has two servers, and has deterministic service time distribution Det(4). At time t both customers begin service, and so at time $t + 4$ both customers finish service. Now Node C has two events that need to be carried out simultaneously. Clearly, as before, the order in which these events take place can affect future waiting times or blockages.

In order to counter any natural bias in the code, whenever more than one independent event must happen simultaneously in Ciw, the order in which they are carried out is randomly generated. First if more than one node is scheduled to have simultaneous events then the node is uniformly randomly chosen. Next, if there are more than one event scheduled simultaneously at that node then these events are uniformly randomly chosen.

This can cause simulations with deterministic time distributions and routing to have stochastic results. As an example, consider the one node network shown in Figure 4.9. This has deterministic inter-arrival times of 10 time units, deterministic service times of 5 time units, 2 parallel servers, and customers always rejoin the queue after service.

Say we are interested in the number of customers who have finished at least one service in a period of 36 time units. Table 4.2 outlines the events that occur between time 0 and time 36, for six different cases that can occur. The bottom row reports on the number of customers N who have finished at least one service in the time period. Six cases occur because at time 30 three events are scheduled to happen simultaneously, customer A rejoining the queue, customer B rejoining the queue, and customer C arriving. The probability tree of the event outcomes is given in Figure 4.10. The six cases for orders of events are:

A) C arrives, A rejoins, B rejoins ($p = 0.25$)

B) C arrives, B rejoins, A rejoins ($p = 0.25$)

C) A rejoins, C arrives, B rejoins ($p = 0.125$)

D) A rejoins, B rejoins, C arrives ($p = 0.125$)

E) B rejoins, C arrives, A rejoins ($p = 0.125$)

F) B rejoins, A rejoins, C arrives ($p = 0.125$)

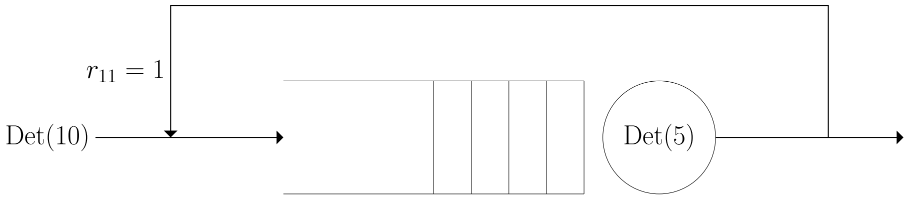


Figure 4.9: A deterministic one node queueing network with certain rejoins.

It can be seen that although the network considered seems completely deterministic, the random choices between simultaneous events can cause stochastic behaviour to

Time	Case A	Case B	Case C	Case D	Case E	Case F
10	A arrives	A arrives	A arrives	A arrives	A arrives	A arrives
15	A rejoins	A rejoins	A rejoins	A rejoins	A rejoins	A rejoins
20	B arrives	B arrives	B arrives	B arrives	B arrives	B arrives
25	A rejoins B rejoins	A rejoins B rejoins	A rejoins B rejoins	A rejoins B rejoins	A rejoins B rejoins	A rejoins B rejoins
30	C arrives	C arrives	A rejoins	A rejoins	B rejoins	B rejoins
30	A rejoins C in service	B rejoins C in service	C arrives	B rejoins	C arrives	A rejoins
30	B rejoins A in service	A rejoins B in service	B rejoins C in service	C arrives	A rejoins C in service	C arrives
35	A & C finish service	B & C finish service	A & C finish service	A & B finish service	B & C finish service	A & B finish service
N	3	3	3	2	3	2

Table 4.2: Timeline of events for the system in Figure 4.9, for the six cases.

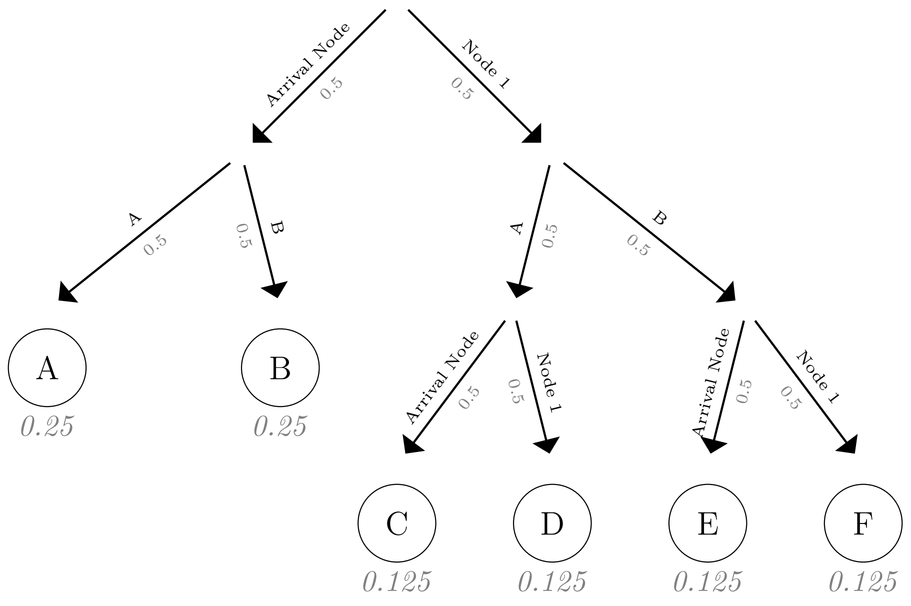


Figure 4.10: Probability tree of event outcomes for the system in Figure 4.9.

occur. In cases D and F only two unique customers have completed a service after 36 time units, whereas in cases A, B, C, and E three unique customers have completed service. From the probability tree shown in Figure 4.10, the probability of only two unique customers having finished a service after 36 time units is $p = 0.25$, and the probability of only three unique customers having finished a service after 36 time units is $q = 0.75$. This understanding allows for increased accuracy in the software's automated testing suite.

4.4 Simultaneous events & arithmetic precision

A curious phenomenon arose while Ciw was being used to model an ophthalmology clinic for Cardiff and Vale University Health Board. This modelling work was carried out by Miss Lieke Hölscher and Dr. Jennifer Morgan. The model was a two node network with entirely deterministic service times. One node was an infinite server node, modelling a delay. The other used a cyclic non-pre-emptive server schedule. A representation of the system is shown in Figure 4.11, where the clocks are used to represent the servers adhere to different shift patterns.

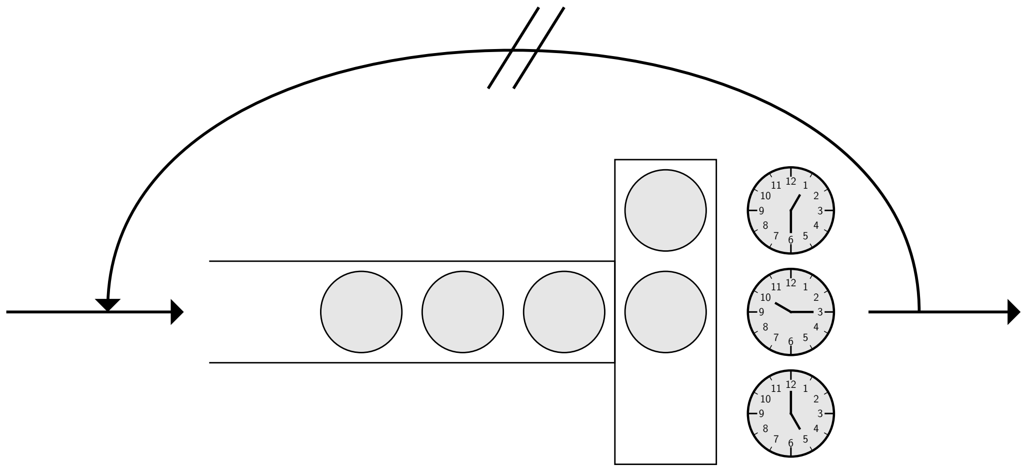


Figure 4.11: Representation of a model of an ophthalmology clinic.

Unexpected behaviour arose as some services were being scheduled to begin at the same time as servers were meant to go off duty, causing simultaneous events. As the service times are deterministic, and the server schedules are cyclic, this behaviour would happen periodically throughout the simulation run.

Recall that from Subsection 4.3, when events are scheduled to take place simultaneously the order is randomly chosen to avoid any bias in the code. Also recall from Subsection 3.3.4 that a server on a non-pre-emptive schedule will always finish its current service before going off duty.

Consider the case, at the end of a server's shift when there are customers waiting, when the 'change shift' event occurs before the 'finish service' event:

```
>>> 0.1 + 0.2
0.30000000000000004
```

(a) Example of floating point arithmetic in Python.

```
>>> from decimal import Decimal
>>> Decimal('0.1') + Decimal('0.2')
Decimal('0.3')
```

(b) Example of Python's `Decimal` type.

Figure 4.12: Comparing Python's floating point arithmetic with the `Decimal` type arithmetic.

1. The server goes off duty, but must finish their service before being killed.

2. A customer finishes service and leaves the server, then the server is killed. A waiting customer cannot begin service as there isn't a server on duty.

Now consider the case, at the end of a server's shift when there are customers waiting, when the 'finish service' event occurs before the 'change shift' event:

1. A customer finishes service and leaves the server. A waiting customer begins service.

2. The server is taken off duty, however they must finish their current service before being killed.

In the second case another service is squeezed into the server's shift, causing that server to work overtime.

Floating point number arithmetic is not precise. Figure 4.12a shows a simple case of this lack of precision. In complex calculations such as in simulation with Ciw, this effect can be cumulative.

Event dates will suffer from this lack of precision. As shift change dates are calculated separately from other event dates (shift changes are drawn from a predefined schedule, whereas other events are scheduled cumulatively after other events), then the 'finish service' event and the 'change shift' event may not be scheduled at exactly the same time, despite them happening simultaneously being the desired outcome.

For example, there may be a ‘change shift’ event scheduled at $t = 4.6$, however the ‘finish service’ event, although logically should be scheduled at the same time, is actually scheduled for $t = 4.5999999999999993$. This causes a bias in which more services are fitted into a server’s shift than should be.

To counter this effect, an ‘exact’ arithmetic option was implemented in Ciw. This makes use of Python’s decimal type instead of the usual float type. An example of this type is shown in Figure 4.12b.

For the model of the ophthalmology clinic, Figure 4.13 shows the parameters used. Figure 4.14 shows the average monthly waiting time for new patients over a period of 100000 days, with and without the exact option. The heatmaps illustrates the average amount of ‘errors’ in 85 runs per five months, where an ‘error’ is indicated as a service beginning at times $t = 0.65$ and $t = 4.51$ in a seven day cycle.

It can be seen that areas where there are discrepancies between the graphs, where the average waiting time halves, correspond to areas that are have high average number of errors. In these areas the errors imply an extra customer being served in a shift. This agrees with the discrepancies in the graphs, where errors correlate with a reduced average monthly waiting time. When using the exact option ‘errors’ are still expected, this is due to random selection on the order of simultaneous events. In fact, as the ‘change shift’ event and the ‘finish service’ event are being chosen uniformly randomly, it would be expected that ‘finish service’ occur before ‘change event’ half the time, thus we would expect roughly half the amount of ‘errors’. This can be seen in the plot.

4.5 Deadlock detection

The deadlock detection method described in Chapter 2 is implemented in Ciw. As shown in Figure 4.2, Ciw has an optional Deadlock Detector component that carries out the procedure. It is constructed in a generic way such that if any other deadlock

```
>>> N = create_network(
...     Number_of_servers=["Inf",
...         [[0, 0.36], [1, 0.65], [0, 4.36], [1, 4.51], [0, 7.0]]],
...     Arrival_distributions={
...         'Class 0': ['NoArrivals', 'NoArrivals'],
...         'Class 1': ['NoArrivals', 'NoArrivals'],
...         'Class 2': ['NoArrivals', 'NoArrivals'],
...         'Class 3': ['NoArrivals', 'NoArrivals'],
...         'Class 4': ['NoArrivals', ['Exponential', 1.288]],
...         'Class 5': ['NoArrivals', 'NoArrivals']},
...     Service_distributions={
...         'Class 0': [['Deterministic', 30],
...                     ['Deterministic', 0.01]],
...         'Class 1': [['Deterministic', 90],
...                     ['Deterministic', 0.01]],
...         'Class 2': [['Deterministic', 180],
...                     ['Deterministic', 0.01]],
...         'Class 3': [['Deterministic', 365],
...                     ['Deterministic', 0.01]],
...         'Class 4': [['Deterministic', 5],
...                     ['Deterministic', 0.01]],
...         'Class 5': [['Deterministic', 5],
...                     ['Deterministic', 0.01]]},
...     Transition_matrices={
...         'Class 0': [[0.0, 1.0,], [1.0, 0.0]],
...         'Class 1': [[0.0, 1.0,], [1.0, 0.0]],
...         'Class 2': [[0.0, 1.0,], [1.0, 0.0]],
...         'Class 3': [[0.0, 1.0,], [1.0, 0.0]],
...         'Class 4': [[0.0, 1.0,], [0.0, 0.0]],
...         'Class 5': [[0.0, 0.0,], [0.0, 0.0]]},
...     Class_change_matrices={
...         'Node 0': [[1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
...                    [0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
...                    [0.0, 0.0, 1.0, 0.0, 0.0, 0.0],
...                    [0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
...                    [0.0, 0.0, 0.0, 0.0, 1.0, 0.0],
...                    [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]],
...         'Node 1': [[0.559, 0.1806, 0.0688, 0.0516, 0.0, 0.14],
...                    [0.1615, 0.3655, 0.2125, 0.1105, 0.0, 0.15],
...                    [0.0891, 0.162, 0.3483, 0.2106, 0.0, 0.19],
...                    [0.09, 0.12, 0.2175, 0.3225, 0.0, 0.25],
...                    [0.1675, 0.1675, 0.1675, 0.1675, 0.0, 0.33]
...                    [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]}
... )
```

Figure 4.13: Parameters used to model an ophthalmology clinic.

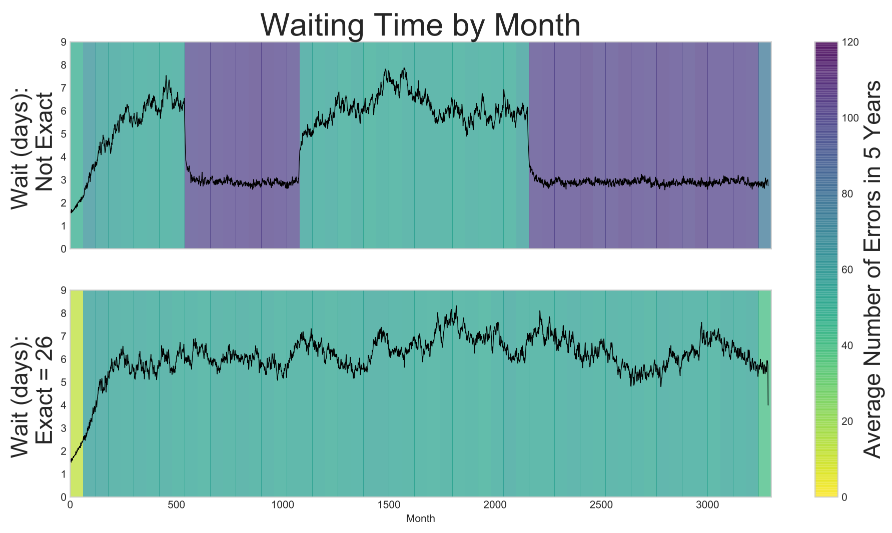


Figure 4.14: Comparing waiting times in ophthalmology clinic with and without exactness (85 runs).

detection methods could be used, they may be coded as a `deadlock_detector` object and will plug in neatly with the Ciw framework. The deadlock detection mechanism described in Chapter 2 is called the State Digraph method.

A generic `deadlock_detector` object is called to perform actions at three specific points in the simulations run: when an server is attached to a customer (that is when a customer begins service), when a server is detached from a customer (that is when a customer leaves a node), and when a blockage occurs. These actions manifest as methods of the `deadlock_detector` object, which manipulate a NetworkX [152] graph object, that represents the state digraph $D(t)$. The actions that the State Digraph method takes at these points is described below:

- At a blockage: An edge is drawn between the representation of that server and the representations of every server at the node which that customer is blocked from entering.

- At server detachment: All in edges to that server's representation are removed from the state digraph.

- At server attachment: If there are any customer who were previously blocked by that server (excluding the current individual beginning service at that server) then those edges are drawn. That is redrawing any edges removed at a detachment, for those customers who are still blocked due to the new customer beginning service.

These three methods are enough to implement the evolution of the state digraph throughout the simulation's run. Finally the `deadlock_detector` object must detect when the system has reached deadlock, that is when the state digraph contains a knot.

A brute force algorithm for finding knots was adapted from the NetworkX developer zone ticket #663 (`https://networkx.lanl.gov/trac/ticket/663`); and is sufficient to identify knots in a directed graph. It uses an implementation of an algorithm to find strongly connected components from the Python package NetworkX. The full adapted knot finding algorithm is incorporated into the `detect_deadlock` method of the Deadlock Detector object. This algorithm is given in Algorithm 1.

The technique loops through all strongly connected components. For each connected component, it loops through each node and checks whether it has any descendants that do not belong to that strongly connected component. If this is true, then that component is a knot. This is equivalent to checking that the number successors of each node is equal to the number of nodes in that strongly connected component, as every node in a strongly connected component is a successor of every other node in that component. All knots of the state digraph as appended to a list. If the list is empty then there is no knots and the system is not in deadlock, else the system has reached deadlock. This method is called every time the simulation clock is moved forward in time.

Algorithm 1: An algorithm for finding knots in the state digraph $D(t)$.

```
input: $D(t)$.
$K = \emptyset$
Find set $C$ of all strongly connected components of $D(t)$.
for subgraph $d \in C$ do
    if $|d| = 1$ then
        Define $v$ as the vertex of $d$.
        if path from $v$ to $v$ then
            Append $d$ to $k$
        end
    end
    else
        for vertex $v \in d$ do
            Find set $S$ of all descendants of $v$
            if $|S| \leq |d|$ then
                Append $d$ to $K$
                break loop
            end
        end
    end
    if $K > 0$ then
        $D(t)$ contains a knot
    end
    else
        $D(t)$ does not contain a knot
    end
end
```

4.6 Summary

The internal mechanics of Ciw are discussed in this chapter. Ciw implements the event scheduling approach, categorising events into scheduled **B**-events and unscheduled **C**-events. In Ciw this is implemented in an object orientated manner.

This method gives rise to the possibility of simultaneous events, which combined with Python's floating point arithmetic can be problematic. This is dealt with by randomly choosing the order in which to carry out simultaneous events, and by using an exact arithmetic option.

Finally the deadlock detection method discussed in Chapter 2 is implemented in Ciw, making use of the NetworkX library for constructing the state digraph and finding knots.

Chapter 5

Markov Models of Deadlock

In this chapter Markov chain models of small open restricted queueing networks are built, in order to investigate the effect of system parameters on the time to deadlock.

The following three networks describe all possible configurations of deadlocking queueing networks with two or fewer nodes:

(i) Open one node, multi-server restricted queueing network with feedback loop.

(ii) Open two node, multi-server restricted queueing network with routes between nodes but no self-loops.

(iii) Open two node, multi-server restricted queueing network with routes between nodes and self-loops.

This chapter is structured as follows:

- Section 5.1 presents the general method of manipulating Markov chains used throughout this chapter.
- In Section 5.2 Markov models are built for network (i), and is used to investigate the system's expected time to deadlock. Extensions are given to incorporate baulking customers and the scheduled vacations of servers.
- In Section 5.3 a Markov model is built for network (ii) and its expected time

to deadlock investigated.

- In Section 5.4 Markov models of a single-server version of network (iii) is given. Its expected time to deadlock is investigated, and its behaviour in the limits of its service times is discussed. A bound is given on the expected time to deadlock of this system in terms of the previous two models.

The state space for the multi-server network (iii) is too large to model in a similar manner to the others, and so isn't considered here.

Sections 5.1 5.3, the beginning of Sections 5.2 and 5.4 of this chapter have been published in [122].

5.1 Analysis of Markov chain models

In general a continuous Markov chain model of a deadlocking queueing network is a set of states S and the transition rates between these states q_{s_1,s_2}. Each state $s \in S$ uniquely defines a configuration of customers around the queueing network. Deadlocked states are also present, those that can be reached at first instance. In this work they will either denoted by that specific configuration of customers (if unique), or by negative numbers, for example -1. By definition, deadlocked states cannot transition to any other state (deadlocks after first instance deadlocks are not considered), thus are absorbing states of the Markov chain. Therefore any queueing network that can experience deadlock is guaranteed to experience deadlock, as absorbing Markov chains are guaranteed to enter one of its absorbing states.

The expected time until deadlock is reached, ω, is equivalent to the expected time to absorption of the Markov chain, which can be found using linear algebraic techniques [157]. The canonical form of a discrete absorbing Markov chain is

$$P = \begin{pmatrix} T & U \\ 0 & I \end{pmatrix}$$

where I is the identity matrix.

Now the expected number of time steps until absorption starting from state i is the ith element of the vector

$$(I - T)^{-1}e \tag{5.1}$$

where e is a vector of 1s.

Therefore by discretising the continuous Markov chain and ensuring the correct order of states, the expected number of time steps to absorption, or deadlock can be found.

When there is more than one deadlocked state, there is more than one absorbing state in the Markov chain. Here the expected time to absorption is the expected time to any of the deadlocked states, whichever one is reached first. The probabilities of which absorbing state a Markov chain will reach are given by the $(i, j)^{\text{th}}$ element of the vector

$$(I - T)^{-1}U \tag{5.2}$$

which corresponds to the probability of reaching absorbing state j from transient state i.

In this chapter continuous time Markov chains are defined, and discretised in order to perform the above analyses.

A concept that is discussed throughout this chapter is the function $\omega(\mu)$. It is the function that describes how the time to deadlock varies with the service rate μ.

This will be discussed by numerical observations and by analysing the processes of the queueing system under consideration.

5.2 One node system

Consider the open one node multi-server restricted queueing network shown in Figure 5.1. This shows an $M/M/c/n$ queue where customers arrive at a rate of Λ and served at a rate μ. Once a customer has finished service they rejoin the queue with probability r_{11}, and so exit the system with probability $1 - r_{11}$.

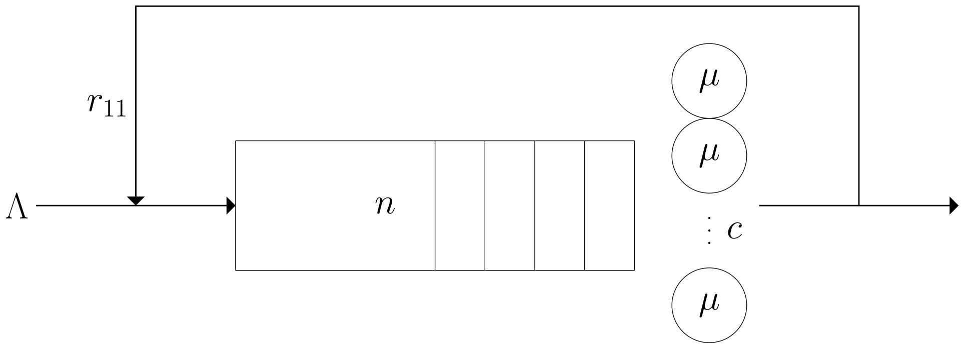


Figure 5.1: An open one node multi-server restricted queueing network.

The state space is given by:

$$S = \{i \in \mathbb{N} \mid 0 \leq i \leq n + 2c\}$$

where i denotes the number of individuals in the system plus the number of individuals who are blocked. For example, $i = n + c + 2$ denotes a full system, $n + c$ individuals in the node, and 2 of those individuals are also blocked. The state $i = n + 2c$ denotes the deadlocked state.

Define $\delta = i_2 - i_1$ for all $i_k \in S$. The transitions are given by Equations 5.3 and 5.4.

$$q_{i_1,i_2} = \left\{ \begin{array}{rl} \Lambda & \text{if } \delta = 1 \\ (1 - r_{11})\mu\min(i, c) & \text{if } \delta = -1 \\ 0 & \text{otherwise} \end{array} \right\} \text{ if } i_1 < n + c \tag{5.3}$$

$$q_{i_1,i_2} = \left\{ \begin{array}{rr} (c-b)r_{11}\mu & \text{if } \delta = 1 \\ (1-r_{11})(c-b)\mu & \text{if } \delta = -b-1 \\ 0 & \text{otherwise} \end{array} \right\} \quad \text{if } i_1 = n+c+b \quad \forall \quad 0 \leq b \leq c \tag{5.4}$$

where b denotes the number of blocked customers. The Markov chain is shown in Figure 5.2.

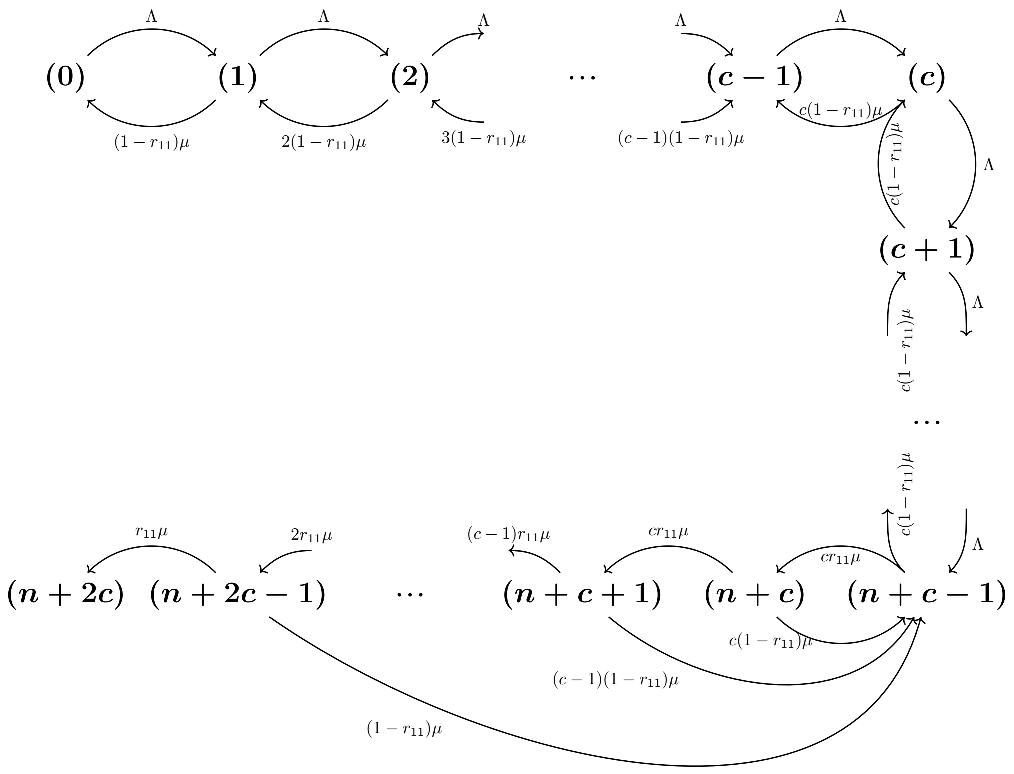
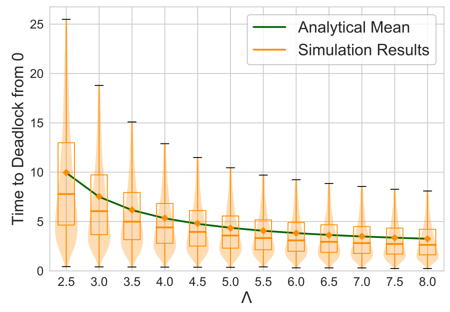


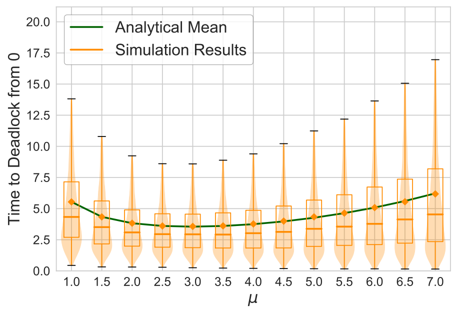
Figure 5.2: Transition diagram of the Markov chain for a multi-server one node system.

Figure 5.3 shows the effect of varying the parameters of the above Markov model. The orange box and violin plots give the distribution of results obtained by the simulation model over 10,000 trials, and the green line plot gives the results obtained by the Markov model. Base parameters of $\Lambda = 6$, $n = 3$, $\mu = 2$, $r_{11} = 0.5$ and $c = 2$ were used.

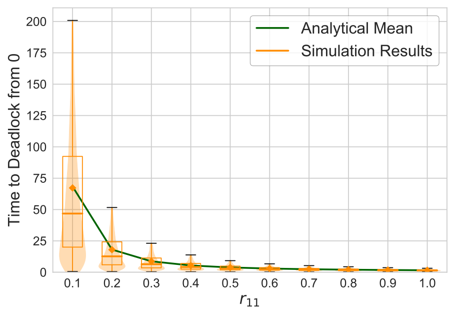
Some observations:



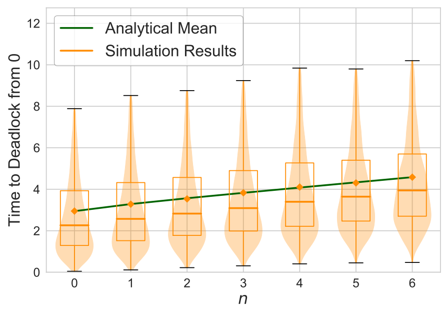
(a) Varying Λ



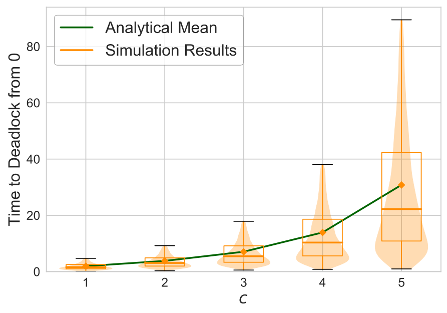
(b) Varying μ



(c) Varying r_{11}



(d) Varying n



(e) Varying c

Figure 5.3: Time to deadlock in the multi-server one node system.

- It can be seen that increasing the arrival rate Λ and the routing probability r_{11} results in reaching deadlock faster. This is intuitive as increasing these parameters results in the queue filling up quicker.

- Increasing the queueing capacity n results in reaching deadlock slower. Again this is intuitive, as increasing the queueing capacity allows more customers in the system before reaching deadlock.

- Increasing the amount of servers c has a similar effect to increasing the queueing capacity, as there are now more transient states to go through before reaching the deadlock state. Varying the amount of servers has a greater effect on the time to deadlock however, as any state in which customers are blocked, $i \in [n + c + 1, n + 2c]$, can jump back to state $i = n + c - 1$ simply with a service where the customer doesn't rejoin the queue. Increasing the amount of servers also increases the rate at which customer leave the system, but not the rate at which customers enter the system. This means that the rate of decrease of the number of customers in the system increases, however the rate of increase of the number of customers in the system does not change, thus it would take longer to reach a full system, a requirement of deadlock.

The behaviour as the service rate μ varies is not monotonic, seen in Figure 5.3b, as the service rate contributes towards both moving customers from the system and allowing customers to rejoin the queue, causing blockages and deadlock. This behaviour is described in the following conjecture.

Conjecture 1. *The function $\omega(\mu)$ that describes the time to deadlock of a one node multi server system as the service rate μ varies, and all other parameters are fixed, has one critical point and is a local minimum for $\mu \in (0, \infty)$.*

The behaviour of $\omega(\mu)$ can be interpreted as follows:

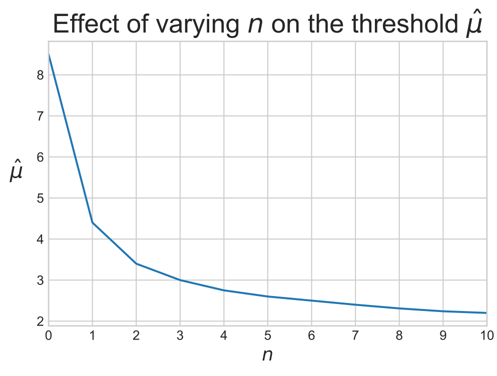
- At $\lim_{\mu \to 0} \omega(\mu)$ the service time approached infinity, and so infinite time until deadlock.

- At low service rates below a certain threshold $\hat{\mu}$, the arrival rate is relatively large compared to the service rate, and a saturated system can be assumed. Now services in which a customer leaves the system does not have much of an effect, as it can be assumed that there will be another arrival promptly. However services in which a customer wishes to rejoin the queue results in a blockage as the system is saturated. Therefore, increasing the service rate increases the chance of a blockage, and so the probability of deadlock.

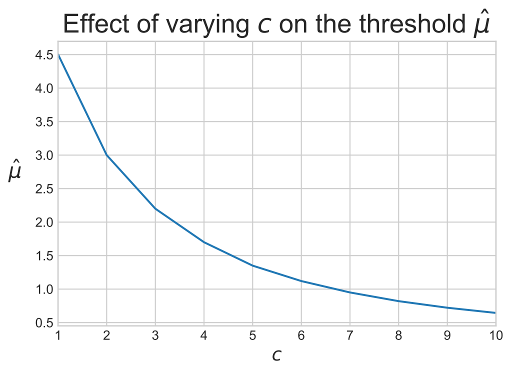
- At high service rates above $\hat{\mu}$, the service rate is large enough in comparison to the arrival rate that a saturated system cannot be assumed, that is a prompt arrival after a customer leaves is unlikely. Thus services in which the customer exits the system does lower the number of customers in the system, and as such there is a smaller probability of getting blocked and deadlocked.

- At $\lim_{\mu \to \infty} \omega(\mu)$ the service time approaches zero, so the queue can never fill up, thus infinite time to deadlock.

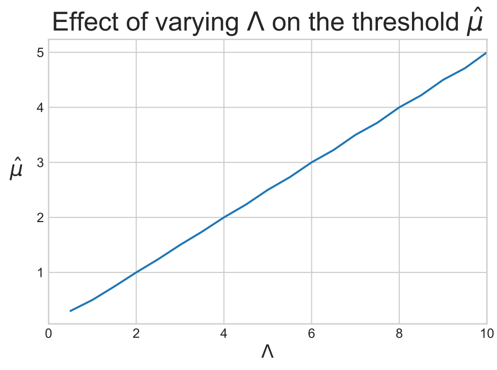
Numerically this service rate threshold $\hat{\mu}$ can be found by computing all $\omega(\mu)$ for all μ, and finding the minimum. Figures 5.4a, 5.4b, 5.4c and 5.4d show the effect of varying n, c, Λ, and r_{11} on $\hat{\mu}$ respectively. Figure 5.4a shows that increasing the queueing capacity lowers the threshold; this is due to the system becoming saturated easier at lower queueing capacities and therefore requiring a larger service rate to escape this saturated zone. Similar behaviour is seen varying c, shown in Figure 5.4b. Figure 5.4c displays a positive linear relationship between the arrival rate and the threshold. At lower arrival rates the system does not fill up as easily, and so is easier to escape the saturated zone. Figure 5.4d shows that increasing the rejoining probability raises the threshold. Lower rejoining probabilities make the system escape the saturated zone quicker as more customers leave the system, so only a low service rate is required to escape; however at high rejoin probabilities a higher service rate fills the system up quicker, and so an even higher service rate is required to escape the saturated zone.



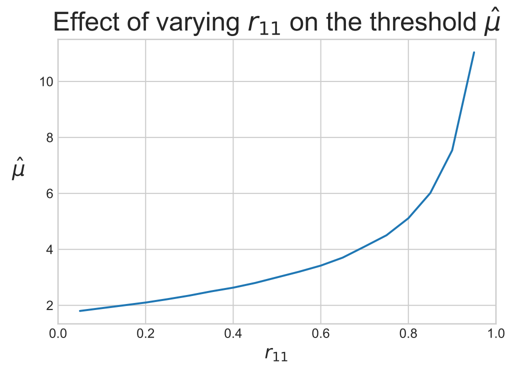
(a) The effect of n on $\hat{\mu}$.



(b) The effect of c on $\hat{\mu}$.



(c) The effect of Λ on $\hat{\mu}$.



(d) The effect of r_{11} on $\hat{\mu}$.

Figure 5.4: Investigating the service rate threshold.

The observation that $\omega(\mu)$ contains one critical point leads to the following conjecture that will be useful in Section 5.4.2.

Conjecture 2. *The function $\omega(\mu)$ attains its maximum over a range at either end point of that range. That is* $\arg\max_{\mu \in [a,b]} \omega(\mu) \in \{a, b\}$.

From the closed interval method of finding absolute maximum [161], the absolute maximum of $\omega(\mu)$ on the closed interval $[a, b]$ is either the critical points in (a, b), a or b. The only critical point in (a, b) is $\hat{\mu}$, and is a local minimum (from Conjecture 1), and so $\omega(\hat{\mu}) \leq \omega(a)$ and $\omega(\hat{\mu}) \leq \omega(b)$. Therefore $\omega(\mu)$ obtains its maximum at either a or b.

The next two subsections contain collaborative work done with Cindy Huang and Dr. Vincent Knight, during Cindy's Nuffield research placement [115] at the School of Mathematics, Cardiff University. At the time, Cindy was a sixth form student undertaking a four week placement with myself and Dr. Knight. The work involved extending the Markov chain model built in this section, in order to model customers' baulking behaviour and servers' scheduled vacations. This gives further insight into the behaviours of the system parameters on the time to deadlock ω.

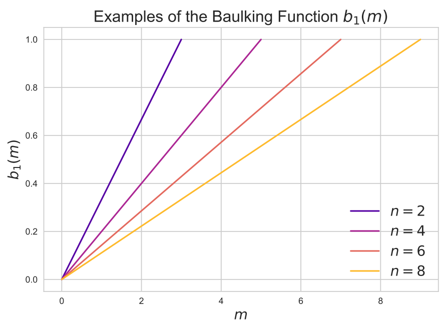
5.2.1 Baulking customers

Baulking behaviour is when customers arrive, but probabilistically choose not to join the queue given the size of the queue. This work investigates two models of baulking discussed in [5] and [6]. Here arriving customers baulk with probability $b(m)$ dependent on m the number of customers already at the node. The two baulking probability density functions considered here are shown in Equation 5.5 and 5.6, introduced in [5] and [6] respectively.

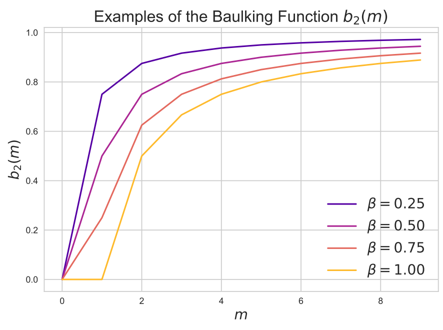
$$b_1(m) = \frac{m}{n+1} \tag{5.5}$$

$$b_2(m) = \begin{cases} 0 & \text{if } m = 0 \\ 1 - \frac{\beta}{m} & \text{otherwise} \end{cases} \tag{5.6}$$

where m is the number of customers already at the node. In Equation 5.5, n denotes the queueing capacity, and in Equation 5.6, β $(0 < \beta \leq 1)$ is some measure of willingness to join the queue. Figure 5.5 shows how the baulking functions $b_1(m)$ and $b_2(m)$ behave for different values of m and β. The function $b_1(m)$ is linear, and enforces a maximum queue capacity of n; the function $b_2(m)$ is not linear, and does not enforce a maximum queue capacity itself. The more customers there are at the node already, the more likely a newly arriving customer is to baulk. For $b_2(m)$, the higher the value of β the less likely a newly arriving customer is to baulk, or the more willing they are to join the queue despite there being customer ahead of them. This justifies the interpretation of β being a measure of willingness to join the queue. When discussing general baulking functions, the index is dropped and is therefore referred to as $b(m)$.



(a) Example of $b_1(m)$, from [5].



(b) Example of $b_2(m)$, from [6].

Figure 5.5: Behaviour of the baulking functions $b_1(m)$ for different capacities n, and $b_2(m)$ for different values of β.

5.2.1.1 Deadlock under $b_1(m)$ baulking

Consider the one node queueing network from Section 5.2 with only one server ($c = 1$), and where customers exhibit $b_1(m)$ baulking. Customers arrive to the system at a rate Λ, but baulk with probability $b_1(m) = \frac{m}{n+1}$ when there are m customers already at the node, and so join the queue with probability $1 - b_1(m)$.

The state space for the corresponding Markov chain is given by:

$$S = \{i \in \mathbb{N} \mid 0 \leq i \leq n+1\} \cup \{(-1)\}$$

where i denotes the number of individuals in service or waiting, and (-1) denotes the deadlocked state.

Define $\delta = i_2 - i_1$ for all $i_k \geq 0,\ k \in \{1, 2\}$. The transitions are given by Equations 5.7, 5.8 and 5.9.

$$q_{i_1,i_2} = \begin{cases} \left.\begin{matrix} \Lambda\frac{n-i_1+1}{n+1} & \text{if } i_1 < n+1 \\ 0 & \text{otherwise} \end{matrix}\right\} & \text{if } \delta = 1 \\ (1-r_{11})\mu & \text{if } \delta = -1 \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

$$q_{i,(-1)} = \begin{cases} r_{11}\mu & \text{if } i = n+1 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

$$q_{-1,i} = 0 \tag{5.9}$$

The Markov chain is shown in Figure 5.6.

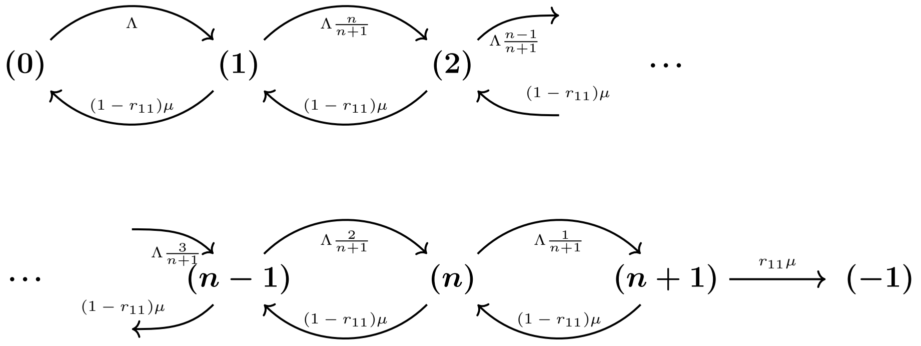

$$\begin{tikzcd}
\mathbf{(0)} \arrow[r, bend left, "\Lambda"] & \mathbf{(1)} \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda\frac{n}{n+1}"] & \mathbf{(2)} \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda\frac{n-1}{n+1}"] & \cdots \arrow[l, bend left, "(1-r_{11})\mu"] & \\
\cdots \arrow[r, bend left, "\Lambda\frac{3}{n+1}"] & \boldsymbol{(n-1)} \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda\frac{2}{n+1}"] & \boldsymbol{(n)} \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda\frac{1}{n+1}"] & \boldsymbol{(n+1)} \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, "r_{11}\mu"] & \mathbf{(-1)}
\end{tikzcd}$$

Figure 5.6: Transition diagram of the Markov chain for the one node single-server system under $b_1(m)$ baulking.

The only difference between this Markov model and the model presented in the previous section when $c = 1$ is that the rate at which customers enter the queue now varies with the number of customers already in the system. The joining rate decreases linearly as the occupancy increases.

To validate the model, Figure 5.7 compares results on the expected time to deadlock from an empty system using this Markov chain, and results obtained from 10,000 simulation trials using Ciw. Base parameters of $\Lambda = 3.0$, $\mu = 3.0$, $r_{11} = 0.6$ were used, while the queueing capacity n varied.

In order to assess the affect of $b_1(m)$ baulking on deadlock, Figure 5.8 shows the

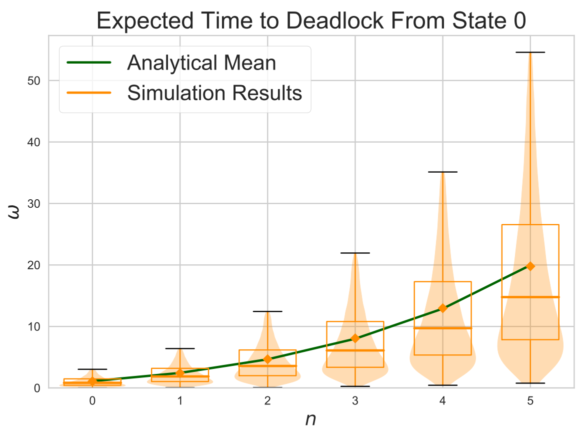


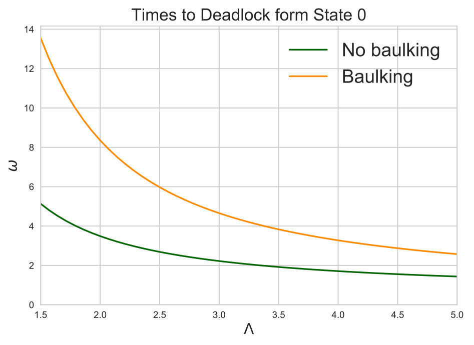
Figure 5.7: Time to deadlock of the one node network under $b_1(m)$ baulking, varying n.

times to deadlock of the system, with and without baulking, as the system parameters vary. Base parameters of $\Lambda = 3.0$, $\mu = 3.0$, $n = 5$, $r_{11} = 0.6$ were used. Under $b_1(m)$ baulking the system takes longer to reach deadlock. This is expected, baulking reduces the joining rate, and as the joining rate decreases, the rate at which occupancy increases decreases, and full occupancy is a requisite for blocking and deadlock. It is also observed that this effect is multiplicative, the longer the time to deadlock, the greater the effect that baulking has on the time to deadlock. Although this isn't true for small values of μ, below the service rate threshold $\hat{\mu}$: this is because below here we assume a saturated system, where between any two services the occupancy fills, and so deadlock can be thought of as independent of the joining rate. The service rate threshold $\hat{\mu}$ however is much lower for the system with baulking. This is because under baulking we would require a larger average service time to ensure that the occupancy fills between services.

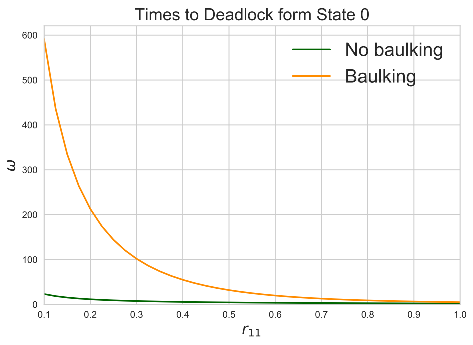
5.2.1.2 Deadlock under $b_2(m)$ baulking

The same system under $b_2(m)$ baulking is now considered. The state space for the corresponding Markov chain is given by:

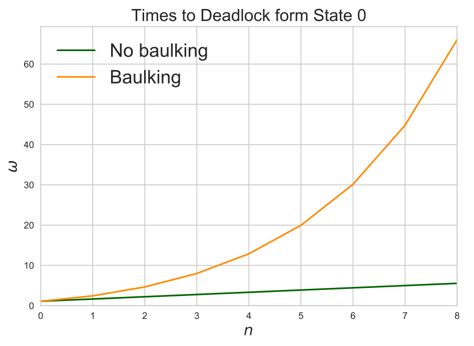
$$S = \{i \in \mathbb{N} \mid 0 \leq i \leq n + 1\} \cup \{(-1)\}$$



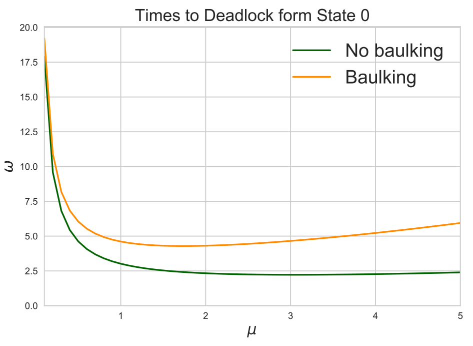
(a) Time to deadlock varying L, with and without $b_1(m)$ baulking.



(b) Time to deadlock varying r_{11}, with and without $b_1(m)$ baulking.



(c) Time to deadlock varying n, with and without $b_1(m)$ baulking.



(d) Time to deadlock varying μ, with and without $b_1(m)$ baulking.

Figure 5.8: The effect of parameters on the time to deadlock; comparing the systems under $b_1(m)$ baulking and without baulking.

where i denotes the number of individuals in service or waiting, and (-1) denotes the deadlocked state.

Define $\delta = i_2 - i_1$ for all $i_k \geq 0,\ k \in \{1, 2\}$. The transitions are given by Equations 5.10, 5.11 and 5.12.

$$q_{i_1,i_2} = \begin{cases} \left.\begin{array}{ll} \Lambda & \text{if } i_1 = 0 \\ \frac{\Lambda\beta}{i_1} & \text{if } 0 < i_1 < n+1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = 1 \\ \hfill (1 - r_{11})\mu & \text{if } \delta = -1 \\ \hfill 0 & \text{otherwise} \end{cases} \tag{5.10}$$

$$q_{i,(-1)} = \begin{cases} r_{11}\mu & \text{if } i = n+1 \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

$$q_{-1,i} = 0 \tag{5.12}$$

The Markov chain is shown in Figure 5.9.

$$\begin{tikzcd}
(\mathbf{0}) \arrow[r, bend left, "\Lambda"] & (\mathbf{1}) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda\beta"] & (\mathbf{2}) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\frac{\Lambda\beta}{2}"] & \cdots \arrow[l, bend left, "(1-r_{11})\mu"] \\
\cdots \arrow[r, bend left, "\frac{\Lambda\beta}{n-2}"] & (\boldsymbol{n-1}) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\frac{\Lambda\beta}{n-1}"] & (\boldsymbol{n}) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\frac{\Lambda\beta}{n}"] & (\boldsymbol{n+1}) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, "r_{11}\mu"] & (\mathbf{-1})
\end{tikzcd}$$

Figure 5.9: Transition diagram of the Markov chain for the one node single-server system under $b_2(m)$ baulking.

If there are m customers in the system then the rate at which customers join the queue is $\frac{\Lambda\beta}{m}$. Note that for any β such that $0 < \beta \leq 1$, and for any number of customers m, there is always a positive probability of a customer joining the queue, and therefore it is always possible to reach the deadlock state. If however $\beta = 0$

then a customer will only join the queue when $m = 0$, and so the system will never reach deadlock, unless $n = 0$.

Figure 5.10 shows the effect of varying the baulking parameter β on times to deadlock, comparing the Markov chain results to simulation results across 10,000 trials. Base parameters of $\Lambda = 3$, $n = 2$, $\mu = 3$, and $r_{11} = 0.6$ are used. As β increases the time to deadlock decreases. As β is interpreted as the willingness of newly arriving customers to join the queue, then this parameter increases more newly arriving customers join the queue, thus the rate at which occupancy increases, increases. The baulking parameter β therefore has a very similar effect on the time to deadlock as the arrival rate Λ.

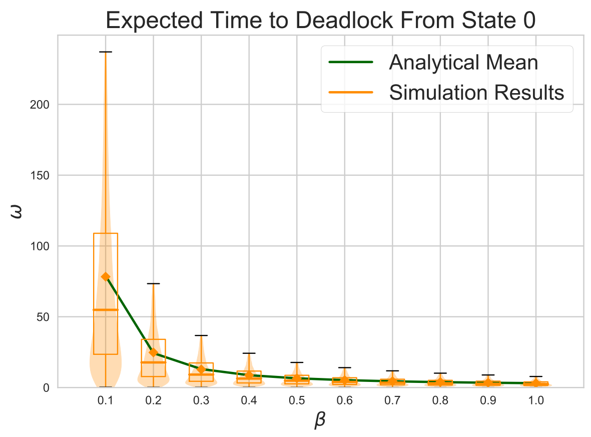
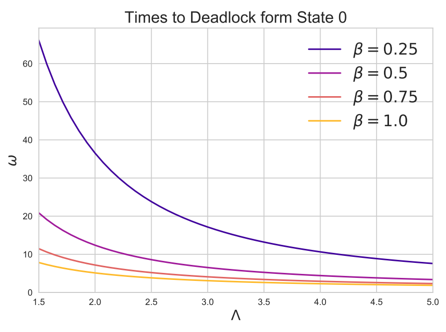


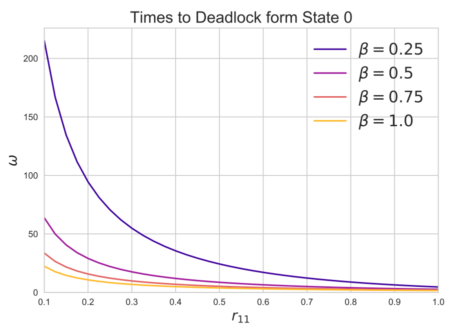
Figure 5.10: Time to deadlock of the one node network under $b_2(m)$ baulking, varying β.

Figure 5.11 shows the effect of the parameters Λ, r_{11}, n and μ on the system, for different values of the baulking parameter β.

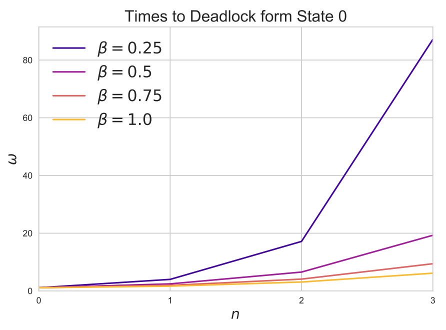
- All three parameters Λ, r_{11}, and β have similar effects on the time to deadlock; as these parameters are increased, then the queueing capacity is filled up more quickly, and so we get to a situation where blocking, and deadlock, can occur sooner. Combining these parameters changes result in an amplified effect.
- Baulking amplifies the effect of increasing the queueing capacity on the time to deadlock, turning an approximately linear effect into an approximately exponential effect. This is because even when customers are most willing to



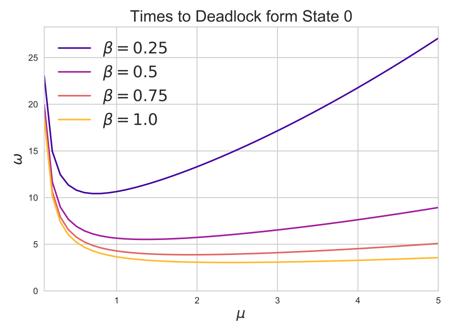
(a) Time to deadlock varying Λ by β.



(b) Time to deadlock varying r_{11} by β.



(c) Time to deadlock varying n by β.



(d) Time to deadlock varying μ by β.

Figure 5.11: Effect of the baulking parameter β on the time to deadlock as parameters change.

join the queue ($\beta = 1.0$) once there are 4 customers in the system three-quarters of arriving customers baulk. Therefore increasing n decreases the likelihood that a customer joins the queue, and so it becomes more and more difficult for capacity to fill, especially when the number of customers already at the queue approaches the queueing capacity.

- The bowl shaped curve is observed in the function $\omega(\mu)$ is still present under baulking. The baulking parameter β effects the shape of the curve: the gradient of the slope for values $\mu > \hat{\mu}$ is steeper at β increases. This is because at service rates lower than $\hat{\mu}$ we assume that the system is saturated, that is the queueing capacity is filled up fast enough that the periods which it is not full are negligible. Therefore in these situations arrivals, and so baulking, cannot affect the time to deadlock. Figure 5.12 shows the effect of the baulking parameter β on the service rate threshold $\hat{\mu}$, showing that the threshold increases as the baulking parameter increases. The baulking parameter affects the amount of newly joining customers, similar to the arrival rate Λ, thus they have a similar effect on $\hat{\mu}$. This relationship is not linear however as in Figure 5.4c, as baulking behaviour is affected by the capacity n, which does not have a linear relationship with $\hat{\mu}$ (Figure 5.4a).

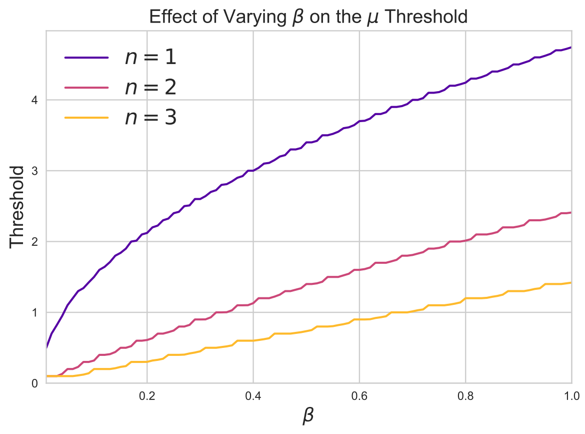


Figure 5.12: The effect of β on the service rate threshold $\hat{\mu}$.

5.2.2 Scheduled vacations

In this section a specific server behaviour is considered. The behaviour that will be focused on is scheduled vacations, that is servers periodically cycling on and off duty. While the server is on duty the system behaves as usual, however when the server is off duty no services can occur, although arrivals happen as usual. At shift changes, pre-emptive interruptions occur, so that when a server goes off duty the current service is interrupted, and resumed when a server comes back on duty.

Let s be the time on duty, and s_v the time on vacation. Therefore the cycle length is $s + s_v$. This type of vacation is categorised in [42] as ‘Maintenance in computer/communications systems (IV)’. Vacations, or maintenance jobs are scheduled periodically every $s + s_v$ time units, and have pre-emptive priority over customers. Maintenance jobs (server vacations) have deterministic service times, each service lasting exactly s_v time units.

As an example, consider an online shop where orders can arrive to a queue at any time of the day or night. The shop hires one server who processes orders from 9am to 5pm, seven days a week. From 5pm to 9am next day, that server is off duty and no orders can be processed, however orders continue to arrive. That server is on a scheduled vacation. In this case s is the time on duty (9am - 5pm), and s_v the time on vacation (5pm - 9am), and the cycle length is one day. Figure 5.13 illustrates this timeline.

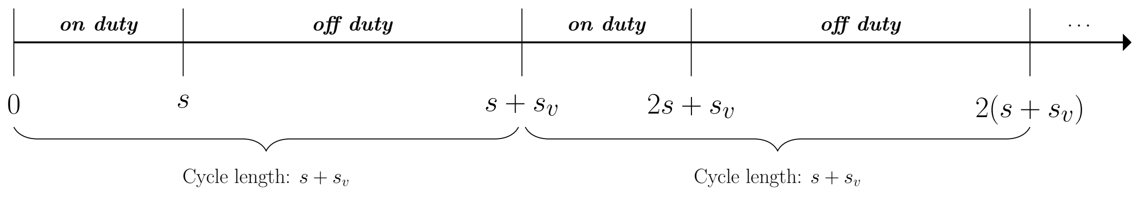


Figure 5.13: Illustration of the (s, s_v) work cycle.

This is modelled using time inhomogeneous Markov chains. First a result on discretising time inhomogeneous Markov chains is required.

5.2.2.1 Discretising time inhomogeneous Markov chains

A well known result [157] on discretising continuous Markov chains gives that the matrix $P = Q\Delta t + I$ is a stochastic matrix (that is the discrete Markov chain P is equivalent to the continuous Markov chain Q with transitions only occurring at the discrete time interval Δt) if the following condition is satisfied:

$$\Delta t \leq \frac{1}{\max_i |q_{ii}|} \tag{5.13}$$

where q_{ij} is the $(i,j)^{\text{th}}$ entry of Q, and I is the identity matrix.

It is also known that if a discrete time inhomogeneous Markov chain can be described by transition matrix P_1 between time steps 1 and 2; and by transition matrix P_2 between time steps 2 and 3, where P_1 and P_2 have the same states, then that system's state can be described by the transition matrix P_1P_2 between time steps 1 and 3. This is equivalent to a discrete time homogeneous Markov chain described by P, being described by P^n for the first n time steps.

Consider a continuous time inhomogeneous Markov chain that is described sequentially by Q_i for s_i time units. In order to discretise this system, the same time step would have to be used for each component Q_i. In addition that time step needs to allow the discretised Q_i's to switch at exactly the required points. Theorem 7 gives the necessary time step.

Theorem 7. *Given a time inhomogeneous Markov chain that is described sequentially by the continuous transition matrices Q_j for s_j time units, $s_j \in \mathbb{Z}$, for $j \in J = \{1, 2, ..., m\}$, then the discretised system can be given by the discretised transition matrices $P_j = Q_j \Delta t + I$, where*

$$\Delta t = \frac{\mathrm{gcd}_j(s_j)}{\left\lceil \frac{\mathrm{gcd}_j(s_j)}{\min_j \bar{\delta}_j} \right\rceil}$$

where $\bar{\delta}_j = \frac{1}{\max_i |q_{ii}|}$, with q_{ii} being the diagonal entries in the transition matrix Q_j.

Proof. The discretised system requires a single time step, Δt. Δt must be chosen such that each continuous transition matrix is discretised using $P_j = Q_j \Delta t + I$, and such that Δt can be multiplied an integer amount of times to give all s_j, for $j \in J$.

First $Q_j \Delta t + I$ is required to be a stochastic matrix for each $j \in J$. In order for this to be true:

$$\Delta t \leq \frac{1}{\max_j |q_{ii}|} \quad \text{for all} \quad j \in J$$

where q_{ii} are the diagonal entries of Q_j. Defining $\bar{\delta}_j = \frac{1}{\max_j |q_{ii}|}$ for all $j \in J$, then:

$$\Delta t \leq \min_j \bar{\delta}_j \tag{5.14}$$

Now $\Delta t \in \mathbb{R}$, but as it is required that Δt can be multiplied by integers to give each of s_j for $j \in J$, this requires $\Delta t \in \mathbb{Q}$. Therefore there exists $k, m \in \mathbb{Z}$ such that $k\Delta t = m$. Now there exists an integer m that divides each s_j for $j \in J$. For efficiency, take the largest such m, and so $m = \text{gcd}_j(s_j)$. Therefore Δt takes the form:

$$\Delta t = \frac{\text{gcd}_j(s_j)}{k} \tag{5.15}$$

Combining the conditions in Equations 5.14 and 5.15, k is obtained such that:

$$\frac{\text{gcd}_j(s_j)}{k} \leq \min_j(\bar{\delta}_j)$$

Rearranging and forcing $k \in \mathbb{Z}$ gives $k = \left\lceil \frac{\text{gcd}_j(s_j)}{\min_j(\bar{\delta}_j)} \right\rceil$. Finally, substituting in for k:

$$\Delta t = \frac{\text{gcd}_j(s_j)}{\left\lceil \frac{\text{gcd}_j(s_j)}{\min_j \bar{\delta}_j} \right\rceil}$$

□

5.2.2.2 Deadlock with scheduled vacations

Now a model can be built for the one node single-server system with scheduled vacations. Customers arrive at a rate of Λ, the queueing capacity is n, and the rejoin probability is r_{11}. The service rate alternates between being μ for s time units, and 0 for s_v time units. If a customer is in service as the service rate changes, their service is interrupted, and is resumed once the server come back on duty after s_v time units on vacation.

This system is described by two transition matrices, Q while the server is on duty, and Q_v when the server is off duty.

The state space for the combined system is:

$$S = \{i \in \mathbb{N} \mid 0 \leq i \leq n+1\} \cup \{(-1)\}$$

where i denotes the number of individuals in service or waiting, and (-1) denotes the deadlocked state. Define $\delta = i_2 - i_1$ for all $i_k \geq 0$.

The transitions for Q are given by Equations 5.16, 5.17 and 5.18.

$$q_{i_1,i_2} = \begin{cases} \left.\begin{array}{ll} \Lambda & \text{if } i_1 < n+1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = 1 \\ (1 - r_{11})\mu & \text{if } \delta = -1 \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

$$q_{i,(-1)} = \begin{cases} r_{11}\mu & \text{if } i = n+1 \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

$$q_{-1,i} = 0 \tag{5.18}$$

The transitions for Q_v are given by Equations 5.19 and 5.20.

$$q_{i_1,i_2} = \begin{cases} \Lambda & \text{if } i_1 < n+1 \text{ and } \delta = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

$$q_{i,-1} = q_{-1,i} = 0 \tag{5.20}$$

These two cases are visualised in Figure 5.14. Note that when a server goes off duty, the capacity for a customer to occupy that space remains, thus there is still space for $n + 1$ customers in the system when the server is off-duty.

Now with Q and Q_v defined, the discretisation time step can be found by using

$$\begin{tikzcd}
(0) \arrow[r, bend left, "\Lambda"] & (1) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda"] & (2) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda"] & \cdots \arrow[l, bend left, "(1-r_{11})\mu"] \\
\cdots \arrow[r, bend left, "\Lambda"] & (n-1) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda"] & (n) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, bend left, "\Lambda"] & (n+1) \arrow[l, bend left, "(1-r_{11})\mu"] \arrow[r, "r_{11}\mu"] & (-1)
\end{tikzcd}$$

(a) Representation of Q, while the server is on duty.

$$\begin{tikzcd}
(0) \arrow[r, bend left, "\Lambda"] & (1) \arrow[r, bend left, "\Lambda"] & (2) \arrow[r, bend left, "\Lambda"] & \cdots \\
\cdots \arrow[r, bend left, "\Lambda"] & (n-1) \arrow[r, bend left, "\Lambda"] & (n) \arrow[r, bend left, "\Lambda"] & (n+1) & (-1)
\end{tikzcd}$$

(b) Representation of Q_v, while the server is off duty.

Figure 5.14: Transition diagram of Q and Q_v, the continuous time Markov chains describing a system with scheduled vacations while the server is on and off duty respectively.

Theorem 7.

Corollary 1. *For the one node single-server restricted network described above, defined by continuous transition matrices Q for s time units and Q_v for s_v time units, the appropriate time step to discretise the transition matrices is:*

$$\Delta t = \frac{\gcd(s, s_v)}{\lceil \gcd(s, s_v) \left(\Lambda + (1 + r_{11}\mu)\right\rceil} \tag{5.21}$$

Proof. From Theorem 7

$$\Delta t = \frac{\gcd_j(s_j)}{\left\lceil \frac{\gcd_j(s_j)}{\min_j \bar{\delta}_j} \right\rceil}$$

Now $\gcd_j(s_j) = \gcd(s, s_v)$. For Q, $\bar{\delta} = \Lambda^{-1}$. For Q_v, $\bar{\delta} = (\Lambda + (1 - r_{11})\mu)^{-1}$. As $(\Lambda + (1 - r_{11})\mu)^{-1} \leq \Lambda^{-1}$, for any $\Lambda \geq 0$, $0 < r_{11} \leq 1$, $\mu \geq 0$, then for any valid parameters $\min_j \bar{\delta}_j = \frac{1}{(\Lambda+(1-r_{11})\mu)}$.

Substituting these into the original theorem yields the required result. □

Two equivalent methods for getting the CDF and expected time to deadlock from these Markov models can be derived using Theorem 7:

1. **Method 1** discretises both matrices Q and Q_v to P and P_v respectively, then uses appropriate matrix multiplication to find the CDF up to some probability $1 - \epsilon$ over time. This makes use of Corollary 1 to find the appropriate discretisaton time step. This is described in Algorithm 2.

2. **Method 2** uses matrix exponential formula $P(t) = e^{Qt}$ [157] for the continuous transition matrices Q and Q_v to find the CDF up to some probability $1 - \epsilon$ over time. This is described in Algorithm 3.

Both methods were attempted. Figure 5.15 shows the CDFs obtained using **Method 1** and **Method 2** for a system with parameters ($\Lambda = 2.0$, $\mu = 4.0$, $r_{11} = 0.2$, $n = 6$,

Algorithm 2: Method 1 of finding the CDF of the time to deadlock of a system with scheduled vacations.

```
input  : $P$, $P_v$, $s$, $s_v$, $\Delta t$, $\epsilon$.
output: CDF of reaching deadlock.
$t, p = 0, 0$
$\hat{P} = P$
$T_{Steps}, CDF = [0], [0]$
while $p < 1 - \epsilon$ do
    $t \leftarrow t + \Delta t$
    if $t \bmod (s + s_v) \leq s$ then
        $\hat{P} \leftarrow \hat{P}P$
    end
    else
        $\hat{P} \leftarrow \hat{P}P_v$
    end
    $p = \hat{P}_{0,-1}$
    Append $p$ to $CDF$
    Append $t$ to $T_{Steps}$
end
$CDF \leftarrow \frac{1}{p}CDF$
return $CDF$, $T_{Steps}$
```

Algorithm 3: Method 2 of finding the CDF of the time to deadlock of a system with scheduled vacations.

```
input  : $Q$, $Q_v$, $s$, $s_v$, $\Delta t$, $\epsilon$.
output: CDF of reaching deadlock.
$t, p = 0, 0$
$\hat{Q} = e^{Q0}$
$T_{Steps}, CDF = [0], [0]$
while $p < 1 - \epsilon$ do
    $t \leftarrow t + \Delta t$
    if $t \bmod (s + s_v) \leq s$ then
        $\hat{Q} \leftarrow \hat{Q}e^{Q\Delta t}$
    end
    else
        $\hat{Q} \leftarrow \hat{Q}e^{Q_v\Delta t}$
    end
    $p = \hat{Q}_{0,-1}$
    Append $p$ to $CDF$
    Append $t$ to $T_{Steps}$
end
$CDF \leftarrow \frac{1}{p}CDF$
return $CDF$, $T_{Steps}$
```

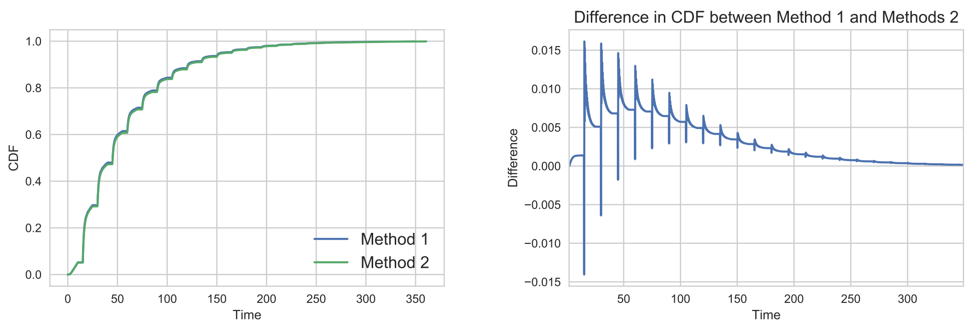


Figure 5.15: Comparison of CDFs obtained using Methods 1 and 2.

$s = 10$, and $s_v = 5$), and the differences in the calculated CDFs.

The results presented from this point onwards are obtained using **Method 1**, in order to be consist with the discretisation methods used in the rest of the chapter. Note also that **Method 1** does not require taking the exponential of a matrix as in **Method 2**, which may indicate it is more reliable [109] in some circumstances.

In order to verify the Markov model, Figure 5.16 shows the time to deadlock of both simulation trials across 10,000 trials, and the analytical expected time to deadlock. Base parameters of $\Lambda = 2.0$, $\mu = 4.0$, $r_{11} = 0.2$, $s = 10$, and $s_v = 5$ were used, for various values of the queueing capacity n. The shape of the time to deadlock distributions highlight the fact that deadlock can only be reached during periods when the server is on duty.

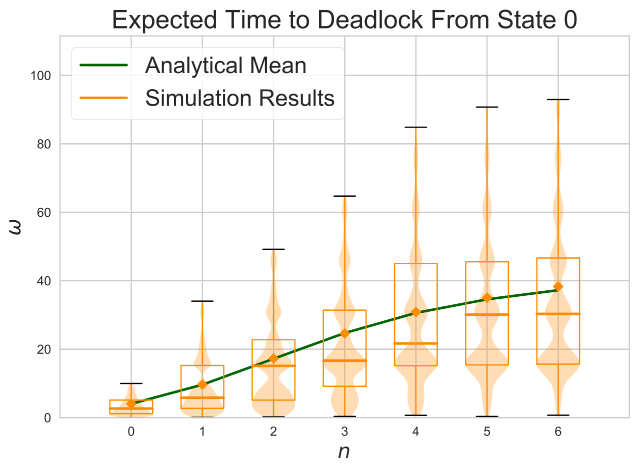


Figure 5.16: Time to deadlock of the one node network with scheduled vacations, varying n.

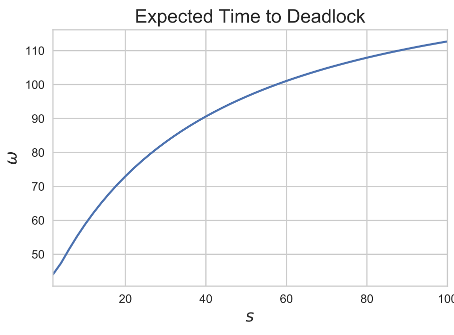
Figure 5.17 shows the effect of the time on duty, s, and the vacation time s_v on the expected time to deadlock ω, and the CDF of reaching deadlock. Now base parameters of $\Lambda = 1.0$, $\mu = 3.0$, $r_{11} = 0.2$, $n = 3$, $s = 25$, and $s_v = 10$ are used.

In order to enable further discussion of the CDF of the time to deadlock, define three periods during the schedule cycle of length $s + s_v$:

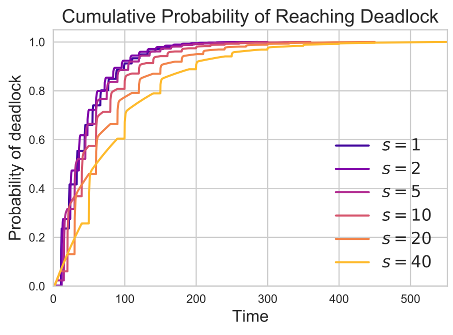
- *'Steady Period'*: during this period arrivals and services occur as usual, and the CDF increases steadily. Note that this does not imply a steady state.
- *'Vacation Period'*: this is the period, lasting s_v time units, where the server is on vacation. During these periods, the CDF remains constant.
- *'Catch-up Period'*: this period occurs immediately after the server returns from vacation, after the system has had a chance to accept arrivals over the Vacation period, but not serve them. During this time the CDF shoots up, and then tails off into the Steady Period. The Steady Period and the Catch-up Period together last s time units.

The increase in the value of the CDF during the Catch-up Period is effected by the amount the system fills up during the Vacation period. The length of the Catch-up Period is effected by how well the server can deal with the build up of customers caused by the vacation. If the server can deal with them well (for example because there is a high service rate) then the Catch-up Period will be short and will swiftly progress to a Steady Period, however if the server fails to cope, then the Catch-up Period will be long, and the system will reach deadlock quicker.

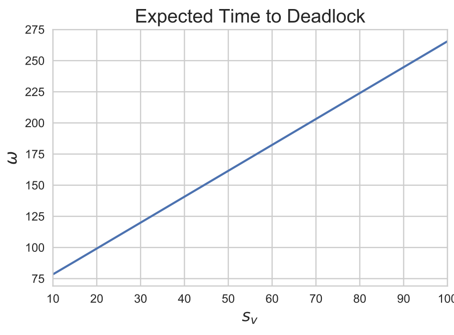
Increasing both s and s_v result in a monotonic increase in the time to deadlock, however the behaviour of the system differs. Increasing s allows a longer Steady Period in which the CDF increases slowly. Therefore at low values of s most of the time the server is on duty accounts for a Catch-up Period, however at high values of s each period where the server is on duty comprises a short Catch-up Period and a longer Steady Period that increases as s increases. As the CDF increases



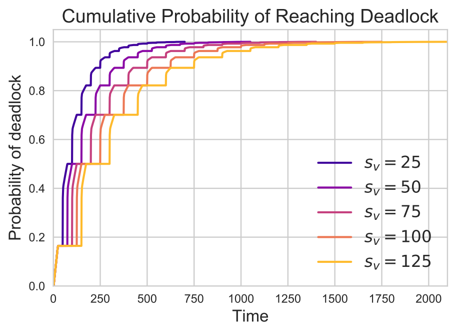
(a) Effect s on ω.



(b) Effect of s on the CDF.



(c) Effect s_v on ω.



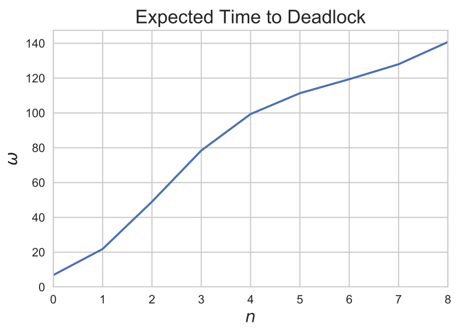
(d) Effect of s_v on the CDF.

Figure 5.17: The effect of s and s_v on ω and the CDF.

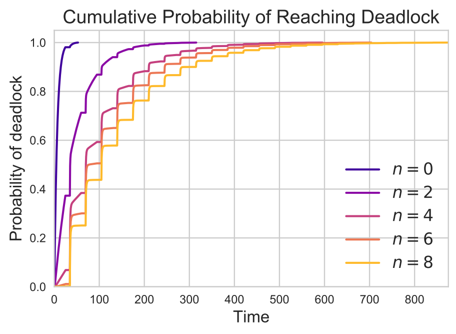
faster during the Catch-up Period, and at lower values of s the system has a greater frequency of Catch-up Periods, then the CDF will increase more quickly, and so the expected time to deadlock is lower. On the other hand, increasing s_v does not seem to have an effect on the height difference of the CDF during the Catch-up Period (probably as the system fills up during the vacation period regardless of how long it is), and so increasing s_v simply delays the CDF from increasing. Therefore higher values of s_v delay the effects of the system reaching deadlock, and so result in a longer time to deadlock.

For the other system parameters, Figure 5.18 give the corresponding plots of ω and the CDF:

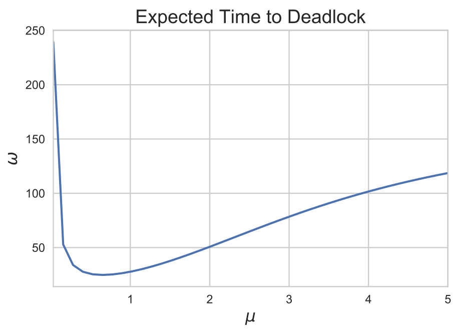
- The height difference of the CDF during the Catch-up Period for smaller values of n is much greater than for larger values. This is intuitive, as a queue with a smaller capacity will have greater chance of filling up during the Vacation Periods than a queue with larger capacity. In addition the increase in CDF during the Steady Periods is steeper for smaller capacities, also intuitive as



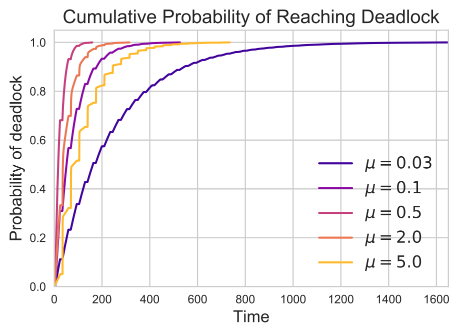
(a) Effect of n on ω.



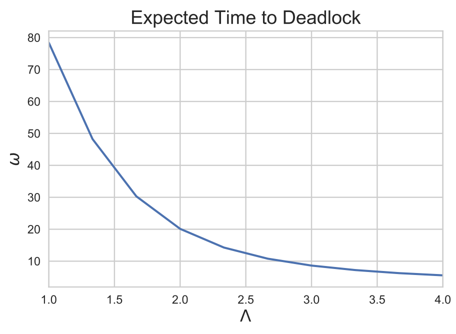
(b) Effect of n on the CDF.



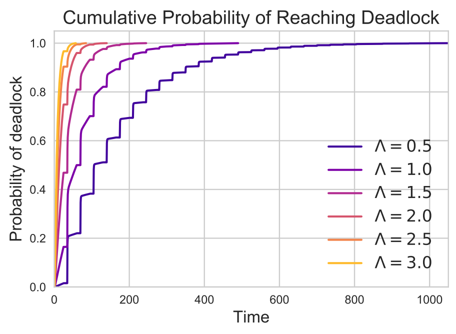
(c) Effect of μ on ω.



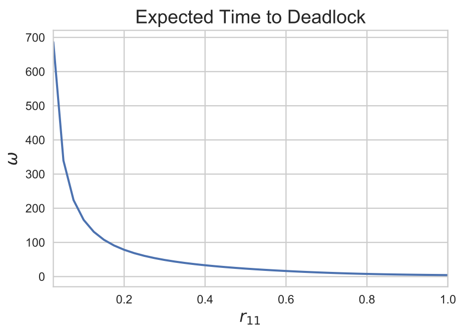
(d) Effect of μ on the CDF.



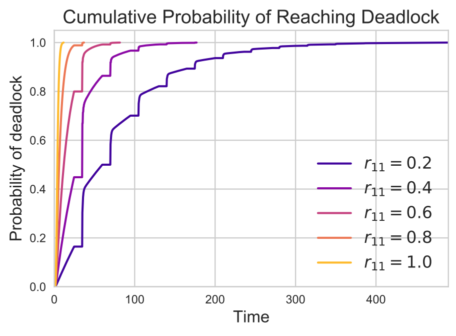
(e) Effect of Λ on ω.



(f) Effect of Λ on the CDF.



(g) Effect of r_{11} on ω.



(h) Effect of r_{11} on the CDF.

Figure 5.18: The effect of system parameters on ω and the CDF for a one node system with scheduled vacations.

smaller capacities can fill up quicker than larger capacities.

- As with the case without scheduled vacations, as $\lim_{\mu \to 0} \omega = \infty$, and $\lim_{\mu \to \infty} \omega = \infty$, with some service time threshold $\hat{\mu}$ where the time to deadlock is minimised. At service rates either side of the threshold, despite having the same expected time to deadlock, the system behaves differently. When $\mu < \hat{\mu}$ the service time is very long, and so it takes a long time for the first customer to finish service after the Vacation Period, and so the Catch-up Period is long and increases slowly. When $\mu > \hat{\mu}$ the service time is short, customers can leave the system quicker, and so the Catch-up Period is shorter and steeper, followed by a Steady Period in which the CDF does not increase much due to customers begin served so quickly.

- Increasing the arrival rate Λ and the transition probability have similar effects, as they both contribute to customers joining the queue. The higher the arrival rate and transition probability the greater the jump in the CFD during the Catch-up Periods. For increasing the arrival rate, this is due to the queue filling up its capacity quicker during the Vacation Period, and there is a higher probability of a fuller queue at the end of the server's vacation. For increasing transition probability however, the arrivals during the Vacation Period stay constant (as there are no services to cause rejoins during this time). The increase in CDF during the Catch-up Period is due to the probability of a rejoin, and hence a blockage and then deadlock, immediately after the Vacation Period, once the queue has had chance to fill. As the transition probability increases this probability increases.

5.3 Two node system without self-loops

Consider the open two node multi-server restricted queueing network shown in Figure 5.19. This shows two $M/M/c_i/n_i$ systems, with service rates μ_i and external arrival rates Λ_i, for each node i. All routing probabilities r_{ij} may be positive apart from self-loops, where $r_{ii} = 0$, for each node i. Note that this system is equivalent to the healthcare systems described in Figure 1.2 and 2.4.

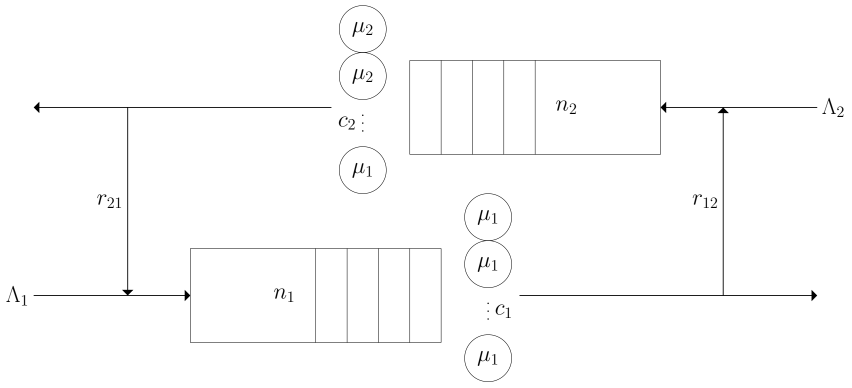


Figure 5.19: An open two node multi-server restricted queueing network.

The state space is given by:

$$S = \{(i, j) \in \mathbb{N}^2 \mid i \leq n_1 + c_1 + j,\ j \leq n_2 + c_2 + i\}$$

where i denotes the number of individuals at node 1 plus the number of individuals blocked waiting to enter node 1, and j denotes the number of individuals at node 2 plus the number of individuals blocked waiting to enter node 2. For example, $(i, j) = (n_1 + c_1 + 2, n_2 + c_2 + 1)$ denotes a full system, $n_1 + c_1$ individuals at node 1, two of whom are blocked waiting to enter node 2; n_2+c_2 individuals at node 2, one of whom is blocked waiting to enter node 1. The state $(i, j) = (n_1+c_1+c_2, n_2+c_2+c_1)$ denotes the deadlock state. The Markov chain is shown in Figure 5.20. The deadlocked state in this case is $(5, 6)$

Define $\delta = (i_2, j_2) - (i_1, j_1)$, $b_1 = \max(0, i_1 - n_1 - c_1)$, $b_2 = \max(0, i_2 - n_2 - c_2)$, $s_1 = \min(i_1, c_1) - b_2$ and $s_2 = \min(i_2, c_2) - b_1$ for all $(i_k, j_k) \in S$. Then the transitions $q_{(i_1,j_1),(i_2,j_2)}$ are given in Table 5.1. The values b_1 and b_2 correspond to the number of customers blocked to node 1 and node 2 respectively. The values s_1 and s_2 correspond to the amount of customers currently in service at node 1 and node 2 respectively.

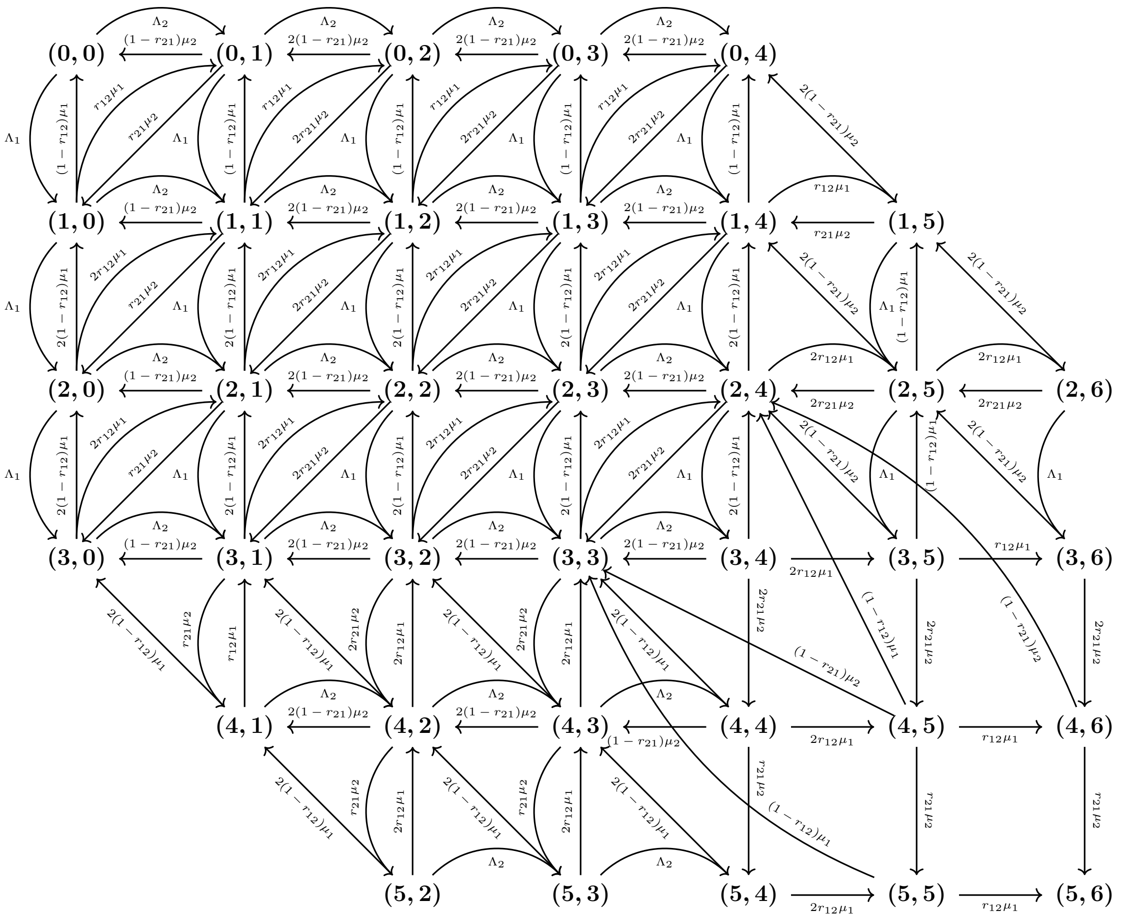
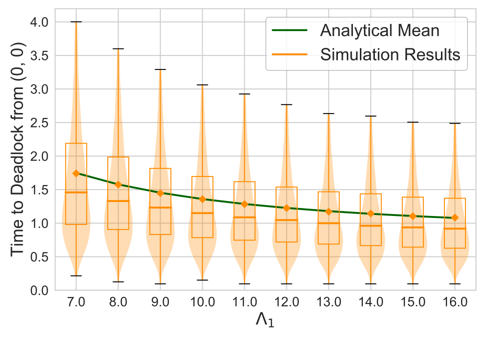


Figure 5.20: Transition diagram of the Markov chain for a multi-server two node system without self-loops with $n_1 = 1$, $n_2 = c_1 = c_2 = 2$.

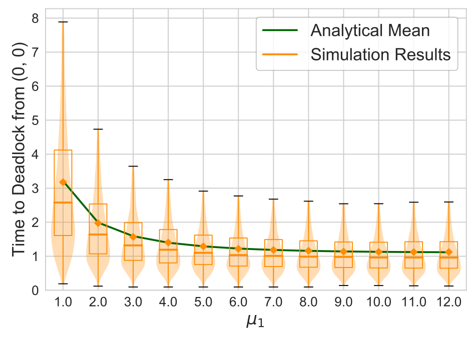
	$j_1 < n_2 + c_2$	$j_1 = n_2 + c_2$	$j_1 > n_2 + c_2$
$i_1 < n_1 + c_1$	Λ_1 if $\delta = (1,0)$ Λ_2 if $\delta = (0,1)$ $r_{12}s_1\mu_1$ if $\delta = (-1,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,-1)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	Λ_1 if $\delta = (1,0)$ $r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,-1)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	Λ_1 if $\delta = (1,0)$ $r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (0,-1)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (-1,-1)$
$i_1 = n_1 + c_1$	Λ_2 if $\delta = (0,1)$ $r_{12}s_1\mu_1$ if $\delta = (-1,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	$r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	$r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,0)$ $(1-r_{21})s_2\mu_2$ if $\delta = (-1,-1)$
$i_1 > n_1 + c_1$	Λ_2 if $\delta = (0,1)$ $r_{12}s_1\mu_1$ if $\delta = (-1,0)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,-1)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	$r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-1,-1)$ $(1-r_{21})s_2\mu_2$ if $\delta = (0,-1)$	$r_{12}s_1\mu_1$ if $\delta = (0,1)$ $r_{21}s_2\mu_2$ if $\delta = (1,0)$ $(1-r_{12})s_1\mu_1$ if $\delta = (-\min(b_1+1, b_2+1), -\min(b_1, b_2+1))$ $(1-r_{21})s_2\mu_2$ if $\delta = (-\min(b_1+1, b_2), -\min(b_1+1, b_2+1))$

Table 5.1: Table of transitions $q_{(i_1,j_1),(i_2,j_2)}$ for a multi-server two node network.

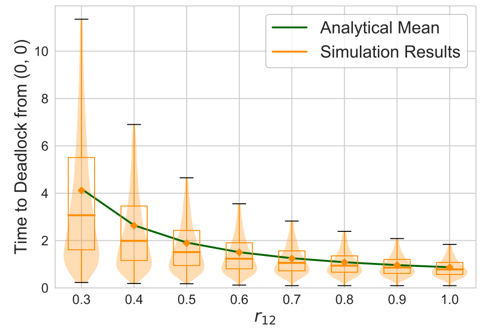
Figure 5.21 shows the effect of varying the parameters of the above Markov model, both analytical results and those from 10,000 simulation trials. Base parameters of $\Lambda_1 = 11.5$, $\Lambda_2 = 9.5$, $n_1 = 2$, $n_2 = 1$, $\mu_1 = 5.5$, $\mu_2 = 6.5$, $r_{12} = 0.7$, $r_{21} = 0.6$, $c_1 = 2$ and $c_2 = 1$ were used. Only plots for the parameters corresponding to node 1 are shown, node 2 shows similar behaviour. Similar behaviour is observed to that seen in Figure 5.3.



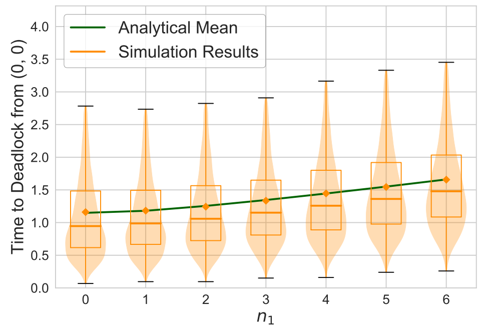
(a) Varying Λ_1



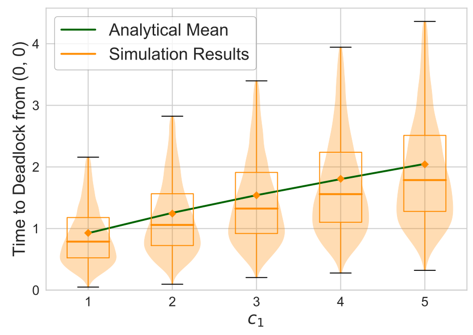
(b) Varying μ_1



(c) Varying r_{12}



(d) Varying n_1



(e) Varying c_1

Figure 5.21: Time to deadlock in a multi-server two node system without self loops.

5.4 Two nodes single-server with self-loops

Consider the open two node single-server restricted queueing network shown in Figure 5.22. This shows two $M/M/1/n_i$ queues with service rates μ_i and external arrival rates Λ_i. All routes are possible, where the routing probability from node i to node j is denoted by r_{ij}. Note that this corresponds to network (iii) from the beginning of this chapter, but with only one server at each node.

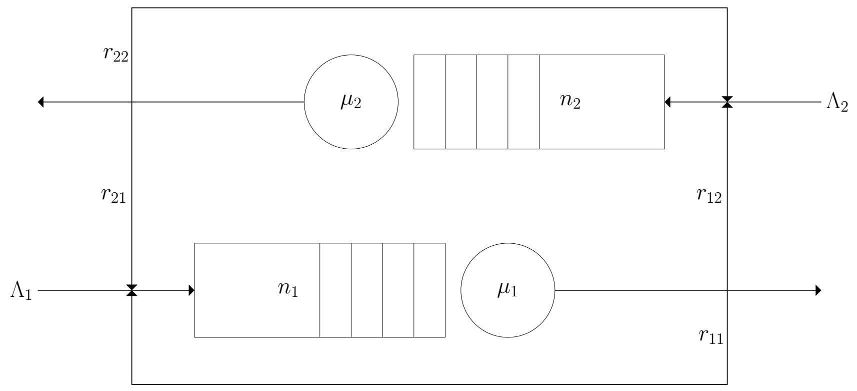


Figure 5.22: An open two node single-server restricted queueing network.

Recall from Theorem 3 that this system will exhibit three different deadlocks that can be reached at first instance (the only deadlocks considered in this chapter). This is a complete network with two nodes, thus there are $S(3,2) = 3$ deadlocks that can be reached at first instance. All three deadlock states must be represented in the state space. Therefore state space is given by:

$$S = \{(i,j) \in \mathbb{N}^2 \mid 0 \leq i + j \leq n_1 + n_2 + 2\} \cup \{(-1), (-2), (-3)\}$$

where i denotes the number of individuals:

- In service or waiting at the first node.
- Occupying a server but having finished service at the second node and waiting to join the first.

where j denotes the number of individuals:

- In service or waiting at the second node.

- Occupying a server but having finished service at the first node and waiting to join the second.

and the state (-3) denotes the deadlock state caused by both nodes; (-1) denotes the deadlock state caused by the first node only; and (-2) denotes the deadlock state caused by the second node only.

Define $\delta = (i_2, j_2) - (i_1, j_1)$ for all $(i_k, j_k) \in S$. The transitions are given by Equations 5.22, 5.23, 5.24, 5.25, and 5.26.

$$q_{(i_1,j_1),(i_2,j_2)} = \begin{cases} \left.\begin{array}{ll} \Lambda_1 & \text{if } i_1 < n_1 + 1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,0) \\ \left.\begin{array}{ll} \Lambda_2 & \text{if } j_1 < n_2 + 1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (0,1) \\ \left.\begin{array}{ll} (1 - r_{11} - r_{12})\mu_1 & \text{if } j_1 < n_2 + 2 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (-1,0) \\ \left.\begin{array}{ll} (1 - r_{21} - r_{22})\mu_2 & \text{if } i_1 < n_1 + 2 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (0,-1) \\ \left.\begin{array}{ll} r_{12}\mu_1 & \text{if } j_1 < n_2 + 2 \text{ and } (i_1, j_1) \neq (n_1 + 2, n_2) \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (-1,1) \\ \left.\begin{array}{ll} r_{21}\mu_2 & \text{if } i_1 < n_1 + 2 \text{ and } (i_1, j_1) \neq (n_1, n_2 + 2) \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,-1) \\ 0 & \text{otherwise} \end{cases} \tag{5.22}$$

$$q_{(i_1,j_1),(-1)} = \begin{cases} r_{11}\mu_1 & \text{if } i > n_1 \text{ and } j < n_2 + 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.23}$$

$$q_{(i_1,j_1),(-2)} = \begin{cases} r_{22}\mu_2 & \text{if } j > n_2 \text{ and } i < n_1 + 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

$$q_{(i_1,j_1),(-3)} = \begin{cases} r_{21}\mu_2 & \text{if } (i,j) = (n_1, n_2+2) \\ r_{12}\mu_1 & \text{if } (i,j) = (n_1+2, n_2) \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

$$q_{-1,s} = q_{-2,s} = q_{-3,s} = 0 \tag{5.26}$$

For $n_1 = 1$ and $n_2 = 2$, the resulting Markov chain is shown in Figure 5.23.

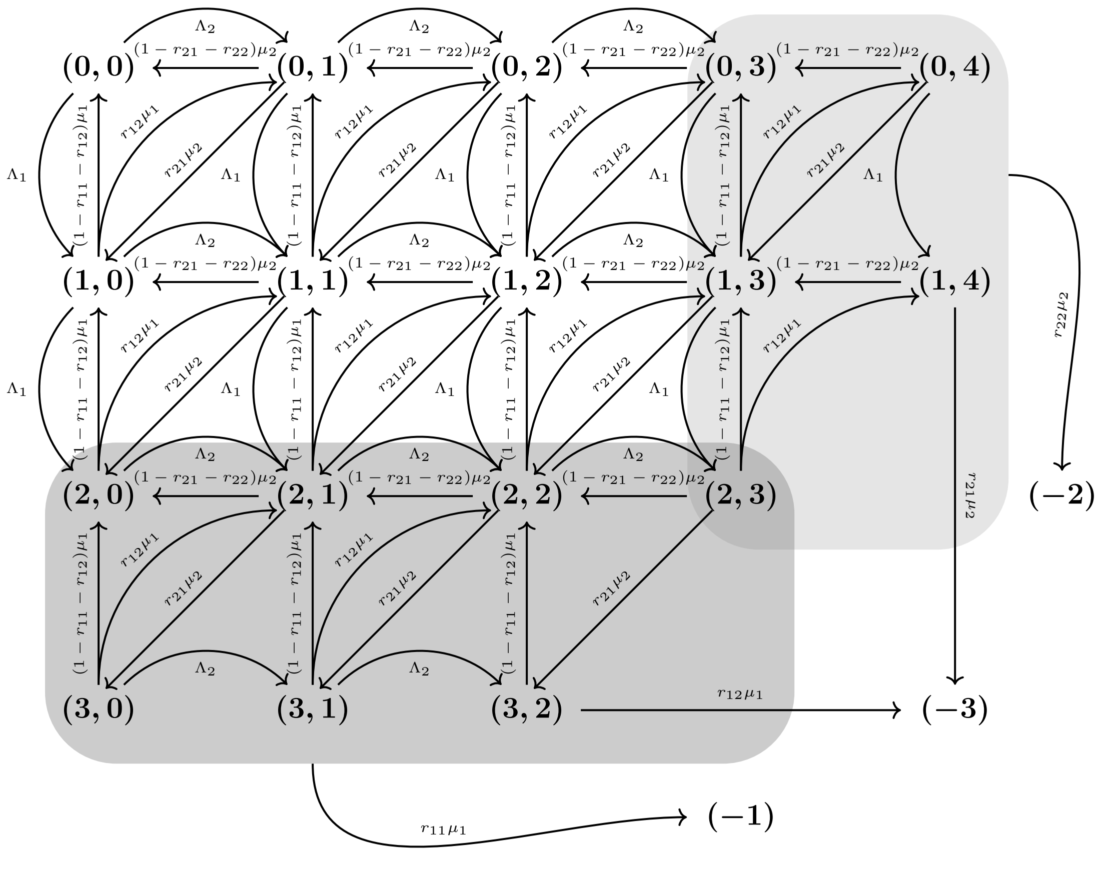
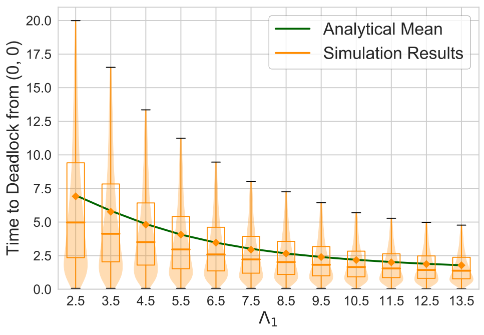
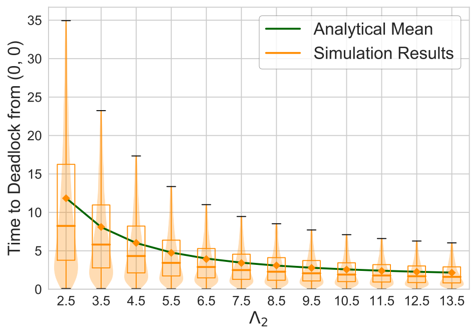


Figure 5.23: Transition diagram of the Markov chain for single server two node system with $n_1 = 1$ and $n_2 = 2$.

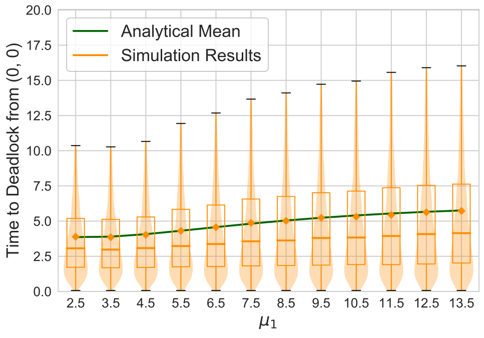
Figure 5.24 shows the effect on the time to deadlock of varying the parameters of the above Markov model, both analytical results and those from 10,000 simulation trials. Base parameters of $\Lambda_1 = 4$, $\Lambda_2 = 5$, $n_1 = 3$, $n_2 = 2$, $\mu_1 = 10$, $\mu_2 = 8$, $r_{11} = 0.1$, $r_{12} = 0.25$, $r_{21} = 0.15$ and $r_{22} = 0.1$ are used.



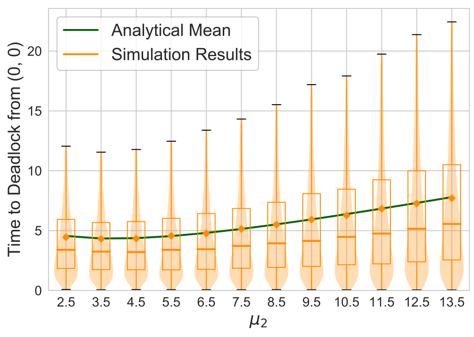
(a) Varying Λ_1



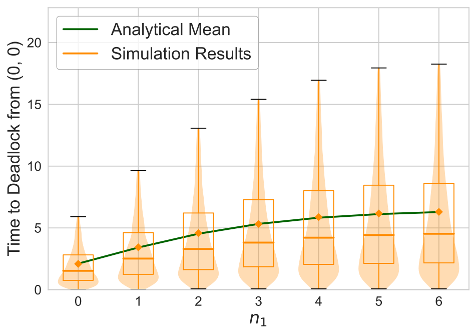
(b) Varying Λ_2



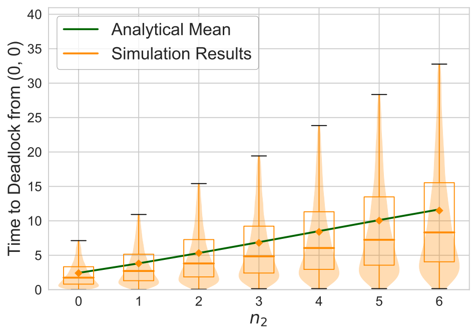
(c) Varying μ_1



(d) Varying μ_2



(e) Varying n_1

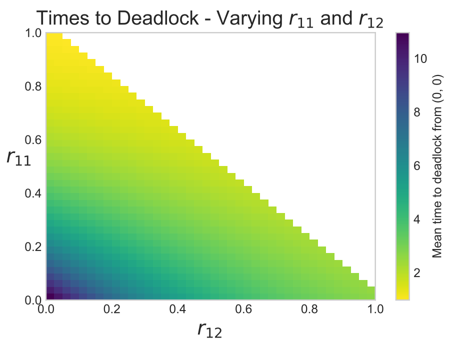


(f) Varying n_2

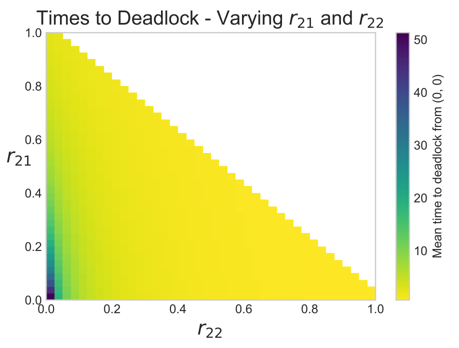
Figure 5.24: Time to deadlock in the single-server two node system.

In general, similar behaviour is observed to that seen in Figures 5.3 and 5.21. A notable difference however is that the increase or decrease in the time to deadlock flattens as the parameter in question increases or decreases. This is clearly observable in Figure 5.24e. This is explained by the existence of more than one deadlock state for this system. Deadlock state (-1) involves node 1 only, deadlock state (-2) involves node 2 only, while deadlock state (-3) involves both nodes. Therefore if a parameter at node 1 is increased or decreased such that the time to a deadlock involving node 1, states (-1) and (-3), approaches infinity, then the overall time to deadlock of the system will become unchanging, as varying that parameter will not effect the time to deadlock state (-2).

The heatmaps in Figure 5.25 illustrate how varying the two transition probabilities out of each node affects the time to deadlock. The shape of the heatmap is due to the restriction that $r_{11} + r_{12} \leq 1$ and $r_{21} + r_{22} \leq 1$. We can see for both nodes it is the rejoining probability (r_{11} and r_{22}) that has the most drastic effect on time to deadlock. This effect is greater for node 2, the node that has the smaller queueing capacity.



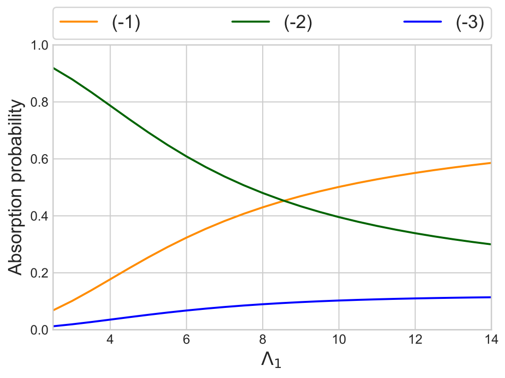
(a) Effect of varying transition probabilities at node 1



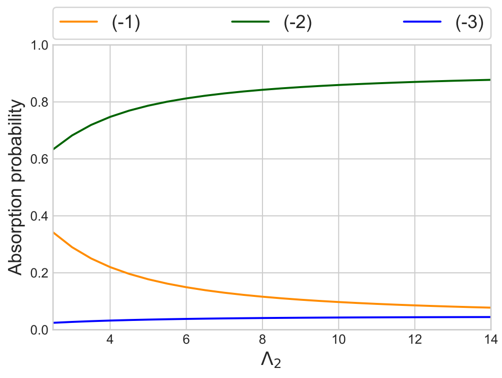
(b) Effect of varying transition probabilities at node 2

Figure 5.25: Effect of the transition probabilities on ω.

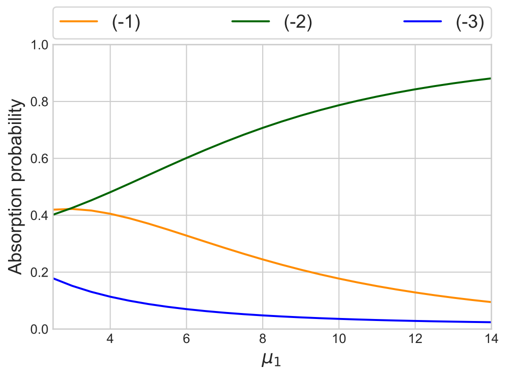
Recall that as there are three deadlock states, the expected time to deadlock ω in all these cases is the expected time to any of the three deadlocks: (-1), (-2) or (-3). Applying Equation 5.2 to this model the probabilities of reaching each of these deadlocked states is found. Figure 5.26 shows how varying the parameters of the queueing network affects the absorption probabilities. In particular Figure 5.26e confirms that the tailing off seen in Figure 5.24e is due to the guaranteed probability of reaching state (-2).



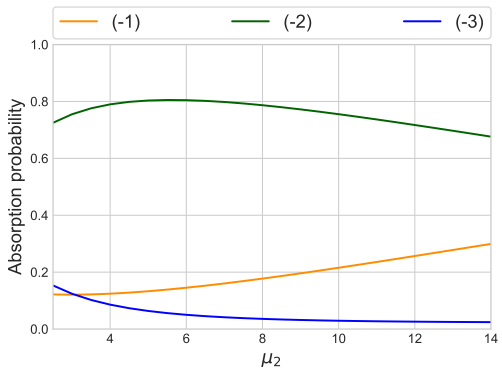
(a) Varying Λ_1



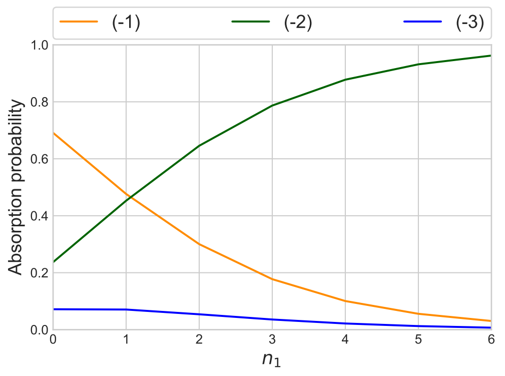
(b) Varying Λ_2



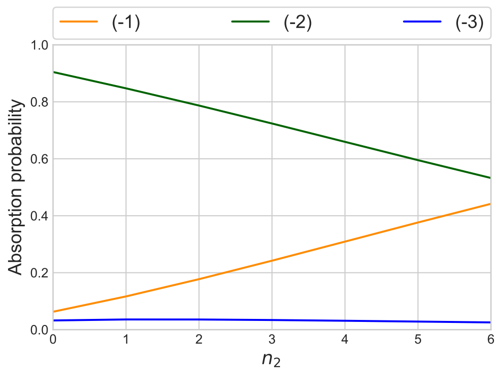
(c) Varying μ_1



(d) Varying μ_2



(e) Varying n_1



(f) Varying n_2

Figure 5.26: Probabilities of reaching each deadlocked state.

5.4.1 Limiting behaviour of $\omega(\mu_1, \mu_2)$

In Section 5.2 is was observed that the function $\omega(\mu)$, the function that describes the expected time to deadlock of a one node system given a service rate μ, has an interesting shape. This is due to the nature of services contributing both to removing customers from the system, and to blockages. This section will consider the function $\omega(\mu_1, \mu_2)$, that describes the time to deadlock of a two node single server system will self-loops, given a service rate μ_1 at node 1 and a service rate μ_2 at node 2. This function also has interesting behaviour. In particular this behaviour is highly dependant on the transition matrix $R = \left(\begin{smallmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{smallmatrix}\right)$.

In order to gain insight into the behaviour of this function, its behaviour in the limits $\mu_1 \to 0$, $\mu_2 \to 0$, $\mu_1 \to \infty$, and $\mu_2 \to \infty$ are analysed. In order to estimate these limits, log-log plots of the function are shown. The following transition matrices will be considered:

- $R_1 = \left(\begin{smallmatrix} r_{11} & 0 \\ 0 & r_{22} \end{smallmatrix}\right)$
- $R_2 = \left(\begin{smallmatrix} r_{11} & 0 \\ 0 & 0 \end{smallmatrix}\right)$
- $R_3 = \left(\begin{smallmatrix} 0 & 0 \\ 0 & r_{22} \end{smallmatrix}\right)$
- $R_4 = \left(\begin{smallmatrix} 0 & r_{12} \\ 0 & r_{22} \end{smallmatrix}\right)$
- $R_5 = \left(\begin{smallmatrix} r_{11} & 0 \\ r_{21} & 0 \end{smallmatrix}\right)$
- $R_6 = \left(\begin{smallmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{smallmatrix}\right)$
- $R_7 = \left(\begin{smallmatrix} r_{11} & 0 \\ r_{21} & r_{22} \end{smallmatrix}\right)$
- $R_8 = \left(\begin{smallmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{smallmatrix}\right)$
- $R_9 = \left(\begin{smallmatrix} 0 & r_{12} \\ r_{21} & r_{22} \end{smallmatrix}\right)$
- $R_{10} = \left(\begin{smallmatrix} r_{11} & r_{12} \\ r_{21} & 0 \end{smallmatrix}\right)$
- $R_{11} = \left(\begin{smallmatrix} r_{11} & r_{12} \\ 0 & 0 \end{smallmatrix}\right)$
- $R_{12} = \left(\begin{smallmatrix} 0 & 0 \\ r_{21} & r_{22} \end{smallmatrix}\right)$
- $R_{13} = \left(\begin{smallmatrix} 0 & r_{12} \\ r_{21} & 0 \end{smallmatrix}\right)$

where $r_{11}, r_{12}, r_{21}, r_{22} > 0$. Note that the remaining combinations, $R = \left(\begin{smallmatrix} 0 & r_{12} \\ 0 & 0 \end{smallmatrix}\right)$, $R = \left(\begin{smallmatrix} 0 & 0 \\ r_{21} & 0 \end{smallmatrix}\right)$, and $R = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ do not result in deadlock at all, as there are no cycles.

Below are four generic observations that will be used to explain the behaviour of $\omega(\mu_1, \mu_2)$ in the limit as the service rate approaches 0 and ∞.

- **Observation 1**: Consider node i with service rate μ_i. As $\mu_i \to \infty$, the service time at node i approaches 0, so the time spent at that node approaches 0, and no queue builds up. Therefore the time to a blockage approaches ∞, and so

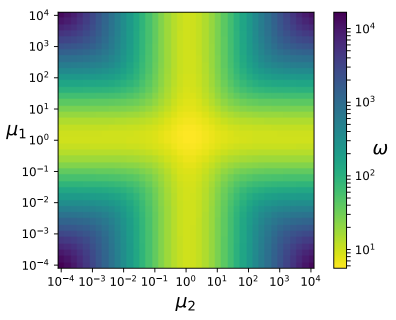


Figure 5.27: Heatmap of $\omega(\mu_1, \mu_2)$ for R_1.

the time to a deadlock involving node i approaches ∞.

- **Observation 2**: Consider node i with service rate μ_i. As $\mu_i \to 0$, the service time at node i approaches ∞, and so no customer can ever leave node i, thus the time until a blockage approaches ∞. Therefore the time until a deadlock involving node i approaches ∞.

- **Observation 3**: Consider nodes i, j with service rates μ_i and μ_j respectively, with $r_{j,i} > 0$. From **Observation 2** as $\mu_i \to 0$ no customer can ever leave node i. So a queue builds up, and a customer from node j may get blocked to node i. This customer will only be unblocked if a customer leaves node i, which cannot happen. Therefore no more customers will leave node j, and by **Observation 2** the time until a deadlock involving node j approaches ∞.

- **Observation 4**: Consider nodes i and j, with $r_{ii} = r_{ij} = 0$. All customers leaving node i also leave the system, and no blockages can occur at that node. Therefore the time to a deadlock involving node i approaches ∞.

Each transition matrix R is now considered:

- $R_1 = \left(\begin{smallmatrix} r_{11} & 0 \\ 0 & r_{22} \end{smallmatrix}\right)$. This is equivalent to two unconnected one node systems, customers cannot transition from one node to the other. Only deadlocks (-1) and (-2) can be reached. The time to deadlock of the whole system is the minimum time to deadlock of the two separate one node systems. From the heatmap in Figure 5.27, the following behaviour is observed at the limits:

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.27}$$

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.28}$$

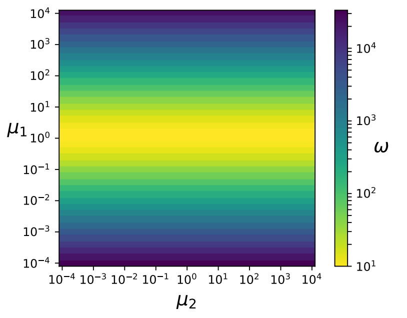
$$\lim_{\mu_1 \to 0} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.29}$$

$$\lim_{\mu_1 \to 0} \lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.30}$$

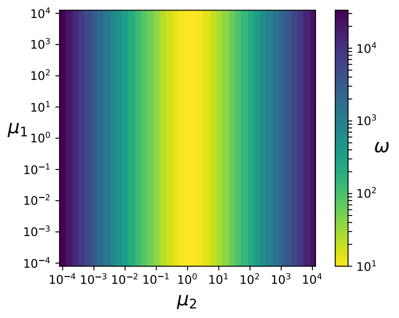
Consider one of these systems. Observations 1 and 2 describe the limiting behaviour of this one node system. Now consider both separate nodes together. The time to deadlock of the system is the minimum time to deadlock of both nodes, and so in order to get an overall divergent time to deadlock, both nodes require divergent times to deadlock. Hence the observed behaviour. Note that μ_1 does not effect the time to deadlock at node 2, nor does μ_2 effect the time to deadlock at node 1.

- $R_2 = \left(\begin{smallmatrix} r_{11} & 0 \\ 0 & 0 \end{smallmatrix}\right)$ and $R_3 = \left(\begin{smallmatrix} 0 & 0 \\ 0 & r_{22} \end{smallmatrix}\right)$. These are equivalent to two one node systems, but only one with a rejoin probability, and customers cannot transfer from one node to the other. Due to Observation 4 the node without a rejoin probability cannot reach deadlock. The time to deadlock of the whole system is the time to deadlock of the one node system with feedback loop. From the heatmaps in Figure 5.28 the following behaviour is observed at the limits for R_2, where μ_2 has no effect on the time to deadlock:

$$\lim_{\mu_1 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.31}$$



(a) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_2.



(b) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_3.

Figure 5.28: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_2 and R_3.

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.32}$$

And the following behaviour is observed for R_3, where μ_1 has no effect on the time to deadlock:

$$\lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.33}$$

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.34}$$

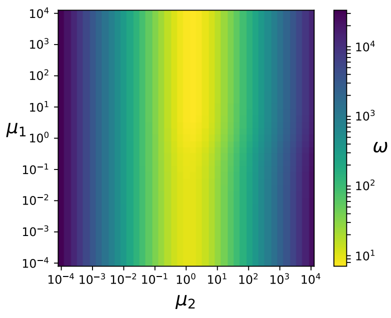
These behaviours are explained by Observations 1 and 2.

- $R_4 = \begin{pmatrix} 0 & r_{12} \\ 0 & r_{22} \end{pmatrix}$ and $R_5 = \begin{pmatrix} r_{11} & 0 \\ r_{21} & 0 \end{pmatrix}$.

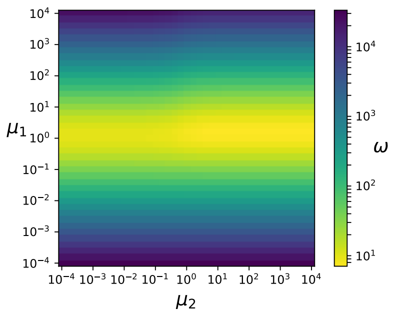
First consider R_4, only the deadlock involving node 2 only can be reached. From the heatmaps in Figure 5.29 the following behaviour is observed at the limits:

$$\lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.35}$$

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.36}$$



(a) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_4.



(b) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_5.

Figure 5.29: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_4 and R_5.

These limits can be explained using Observations 1 and 2. In this system however, the service rate μ_1 does have an effect on $\omega(\mu_1, \mu_2)$, but not at the limits. Increasing μ_1 causes node 2 to have greater joining rate, increasing the time until deadlock.

Similarly for R_5 the following behaviour is observed in the limits:

$$\lim_{\mu_1 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.37}$$

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.38}$$

- $R_6 = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix}$. From the heatmaps shown in Figure 5.30 the following behaviour is observed at following limits:

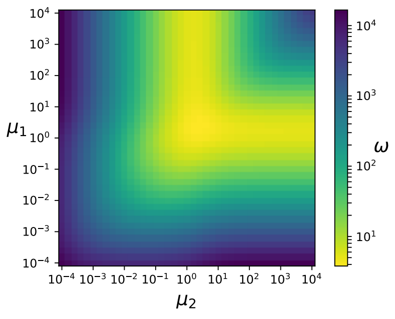


Figure 5.30: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_6.

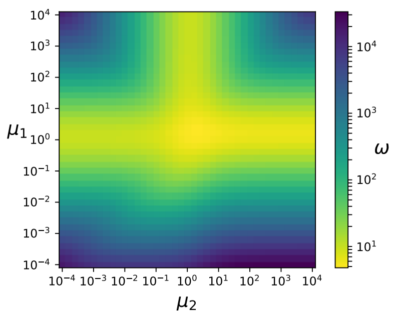
$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.39}$$

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.40}$$

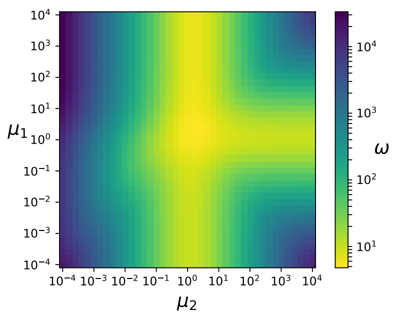
$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.41}$$

As both μ_1 and μ_2 approach infinity together the time to deadlock diverges to ∞. This is intuitive by Observation 1. Observations 2 and 3 explain the limits as the service rates here approach 0.

- $R_7 = \left(\begin{smallmatrix} r_{11} & 0 \\ r_{21} & r_{22} \end{smallmatrix}\right)$ and $R_8 = \left(\begin{smallmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{smallmatrix}\right)$. First consider R_7, its behaviour is a cross between the behaviour of R_1 and R_6. From the heatmaps shown in Figure 5.31 the following behaviour is observed at the limits:



(a) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_7.



(b) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_8.

Figure 5.31: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_7 and R_8.

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.42}$$

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.43}$$

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.44}$$

As both μ_1 and μ_2 approach infinity the time to deadlock diverges to ∞, due to Observation 1. As μ_1 approaches infinity and μ_2 approaches 0, then the time to deadlock diverges to ∞, due to Observations 1 and 2. Finally as μ_1 approaches 0, then the time to deadlock diverges to ∞, due to Observation 3.

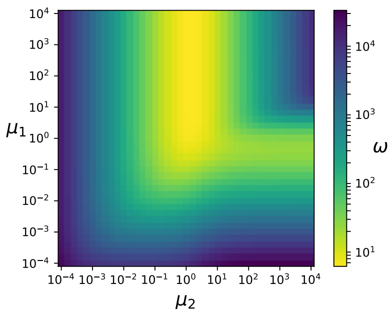
Similarly, for R_8 the following behaviour is observed at the limits:

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.45}$$

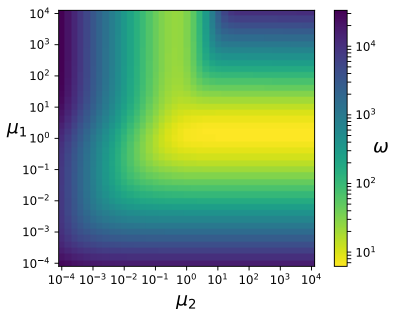
$$\lim_{\mu_1 \to 0} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.46}$$

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.47}$$

- $R_9 = \begin{pmatrix} 0 & r_{12} \\ r_{21} & r_{22} \end{pmatrix}$ and $R_{10} = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & 0 \end{pmatrix}$ First consider R_9, only two deadlocks that can be reached, both involving node 2. From the heatmaps in Figure 5.32 the following behaviour is observed at the limits:



(a) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_9.



(b) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_{10}.

Figure 5.32: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_9 and R_{10}.

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.48}$$

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.49}$$

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.50}$$

As both $\mu_1 \to \infty$ and $\mu_2 \to \infty$ together, then $\omega(\mu_1, \mu_2) \to \infty$, which is due to Observation 1. As $\mu_2 \to 1$ and $\mu_2 \to 0$ then $\omega(\mu_1, \mu_2) \to \infty$, this is due to Observations 2 and 3.

Similarly for R_{10}:

$$\lim_{\mu_1 \to \infty} \lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.51}$$

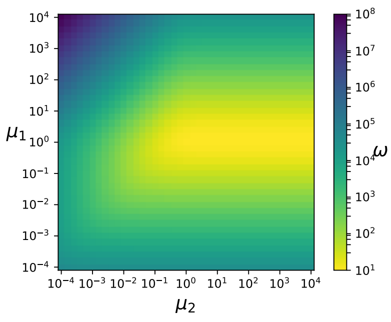
$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.52}$$

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.53}$$

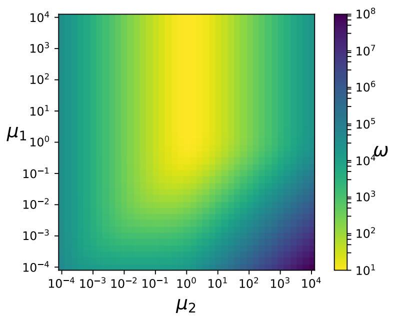
- $R_{11} = \left(\begin{smallmatrix} r_{11} & r_{12} \\ 0 & 0 \end{smallmatrix}\right)$ and $R_{12} = \left(\begin{smallmatrix} 0 & 0 \\ r_{21} & r_{22} \end{smallmatrix}\right)$. First consider R_{11}, here all customer finishing service at node 2 leave the system, so by Observation 4 only one deadlock is possible, the deadlock involving Node 1 only. From the heatmaps shown in Figure 5.33, the following behaviour is observed at the limits:

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.54}$$

$$\lim_{\mu_1 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.55}$$



(a) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_{11}.



(b) Heatmaps of $\omega(\mu_1, \mu_2)$ for R_{12}.

Figure 5.33: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_{11} and R_{12}.

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.56}$$

As either $\mu_1 \to \infty$ or $\mu_1 \to 0$, the time until deadlock approaches ∞, due to Observations 2 and 3. As μ_2 approaches 0 Observation 3 gives that the time to deadlock involving node 2 approaches ∞. As only deadlocks involving node 2 can occur, then the time to deadlock of the system also approaches ∞.

Similarly for R_{10}:

$$\lim_{\mu_1 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.57}$$

$$\lim_{\mu_2 \to \infty} \omega(\mu_1, \mu_2) = \infty \tag{5.58}$$

$$\lim_{\mu_2 \to 0} \omega(\mu_1, \mu_2) = \infty \tag{5.59}$$

- $R_{13} = \begin{pmatrix} 0 & r_{12} \\ r_{21} & 0 \end{pmatrix}$. This system is equivalent to the system described in Section 5.3, in which only one deadlock can be reached. From the heatmaps shown in Figure 5.34, the following behaviour is observed at the limits:

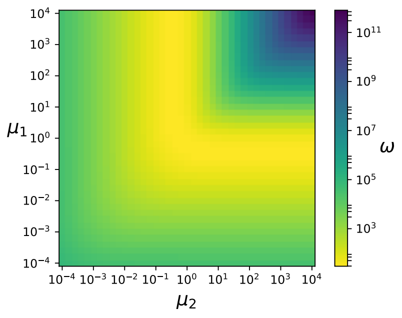


Figure 5.34: Heatmaps of $\omega(\mu_1, \mu_2)$ for R_{13}.

$$\lim_{\mu_1\to\infty}\lim_{\mu_2\to\infty}\omega(\mu_1,\mu_2)=\infty \tag{5.60}$$

$$\lim_{\mu_1\to 0}\omega(\mu_1,\mu_2)=\infty \tag{5.61}$$

$$\lim_{\mu_2\to 0}\omega(\mu_1,\mu_2)=\infty \tag{5.62}$$

Observation 1 explains the limit shown in Equation 5.60, and Observations 2 and 3 explain the limits shown in Equations 5.61 and 5.62.

5.4.2 A bound on the time to deadlock

This section gives a bound on the mean time to deadlock of the system described in Section 5.4. In order to derive this bound, six deadlocking queueing networks are defined as follows, all single-server:

- Define $\Omega^{\star}_{1_1}$ as the 1 node queueing network described in Section 5.2 with the parameter set $(\Lambda_1,\ \mu_1,\ n_1,\ r_{11})$.

  Let its mean time to deadlock be denoted by $\omega^{\star}_{1_1}$.

- Define $\Omega^{\star\star}_{1_1}$ as the 1 node queueing network described in Section 5.2 with the parameter set $(\Lambda_1,\ m_1,\ n_1,\ r_{11})$.

Let its mean time to deadlock be denoted by $\omega^{\star\star}_{1_1}$.

- Define $\Omega^{\star}_{1_2}$ as the 1 node queueing network described in Section 5.2 with the parameter set (Λ_2, μ_2, n_2, r_{22}).

  Let its mean time to deadlock be denoted by $\omega^{\star}_{1_2}$.

- Define $\Omega^{\star\star}_{1_2}$ as the 1 node queueing network described in Section 5.2 with the parameter set (Λ_2, m_2, n_2, r_{22}).

  Let its mean time to deadlock be denoted by $\omega^{\star\star}_{1_2}$.

- Define Ω_2 as the 2 node queueing network described in Section 5.3 with the parameter set (Λ_1, Λ_2, μ_1, μ_2, n_1, n_2, r_{12}, r_{21}). Let its mean time to deadlock be denoted by ω_2.

- Define Ω as the 2 node queueing network described in Subsection 5.4 with the parameter set (Λ_1, Λ_2, μ_1, μ_2, n_1, n_2, r_{11}, r_{12}, r_{21}, r_{22}).

  Let its mean time to deadlock be denoted by ω.

where $m_1 = \frac{\mu_1\mu_2^2}{(\mu_1+\mu_2)^2}$, and $m_2 = \frac{\mu_1^2\mu_2}{(\mu_1+\mu_2)^2}$.

Figure 5.35 shows how Ω contains, and is made up by, Ω_{1_1}, Ω_{1_2} and Ω_2.

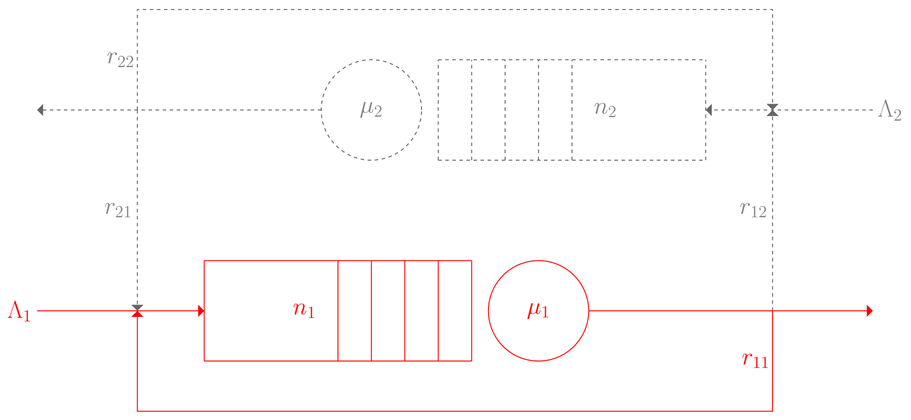
Defining $\omega_{1_j} = \max(\omega^{\star}_{1_j}, \omega^{\star\star}_{1_j})$ for $j \in [1, 2]$, the following theorem is presented giving a bound, assuming that Conjectures 1 and 2 are true:

Theorem 8. *For any valid parameter sets described above, the following inequality holds:* $\omega \leq \min(\omega_{1_1}, \omega_{1_2}, \omega_2)$

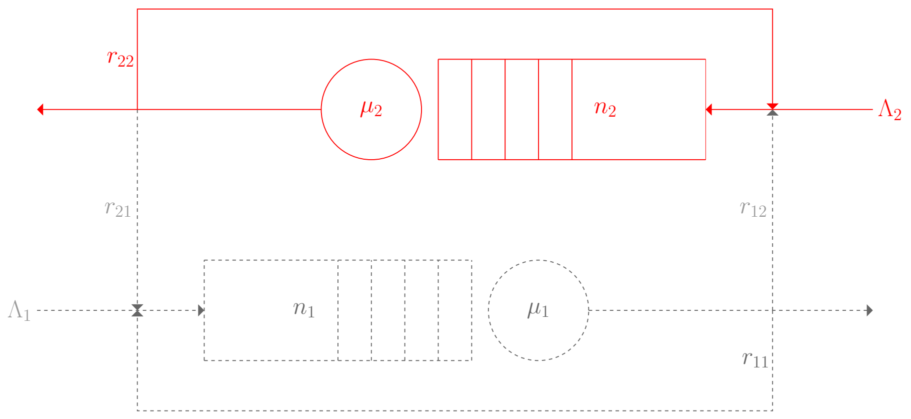
Proof. First, define the following systems:

- Let $\widetilde{\Omega}_{1_1}$ denote the Ω_{1_1} system embedded within Ω. Let $\tilde{\omega}_{1_1}$ denote the mean time to deadlock of $\widetilde{\Omega}_{1_1}$.

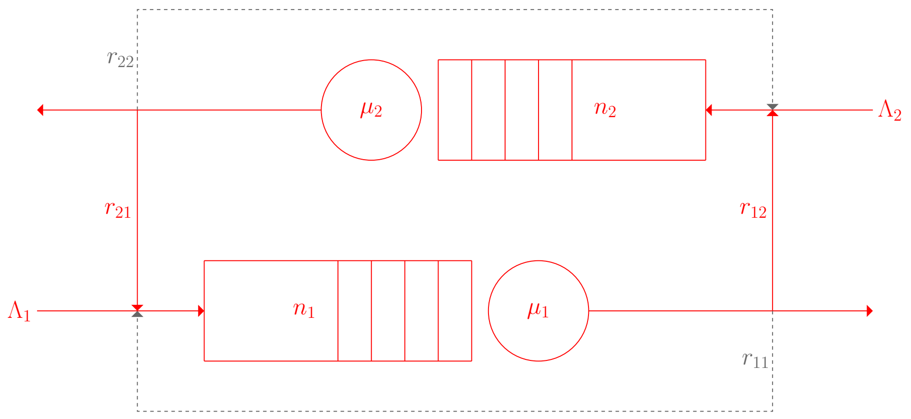
- Let $\widetilde{\Omega}_{1_2}$ denote the Ω_{1_2} system embedded within Ω. Let $\tilde{\omega}_{1_2}$ denote the mean time to deadlock of $\widetilde{\Omega}_{1_2}$.



(a) Ω_{1_1} within Ω



(b) Ω_{1_2} within Ω



(c) Ω_2 within Ω

Figure 5.35: Decomposition of Ω into Ω_{1_1}, Ω_{1_2} and Ω_2

- Let $\widetilde{\Omega}_2$ denote the Ω_2 system embedded within Ω. Let $\widetilde{\omega}_2$ denote the mean time to deadlock of $\widetilde{\Omega}_2$.

It follows that $\widetilde{\omega}_{1_1}$ is the mean time to state (-1) in Ω, $\widetilde{\omega}_{1_2}$ is the mean time to state (-2) in Ω and $\widetilde{\omega}_2$ is the mean time to state (-3) in Ω. Therefore $\omega = \min(\widetilde{\omega}_{1_1}, \widetilde{\omega}_{1_2}, \widetilde{\omega}_2)$, as the mean time to deadlock in Ω is the expected time it takes to reach either (-1), (-2) or (-3).

This proof compares each embedded system with its respective non-embedded counterpart.

1. Consider $\widetilde{\Omega}_2$. The effective arrival rate to node 1 in $\widetilde{\Omega}_2$ is greater than or equal to the effective arrival rate to node 1 in Ω_2, due to the extra customers who are rejoining the queue after service. Similarly the effective arrival rate to node 2 in $\widetilde{\Omega}_2$ is greater than or equal to the effective arrival rate to node 2 in Ω_2. As an increase in the arrival rate causes the mean time to deadlock to decrease, then $\widetilde{\omega}_2 \leq \omega_2$.

2. Consider $\widetilde{\Omega}_{1_1}$. Both the service rate and arrival rate differ in the embedded system to the non-embedded system.

   - Consider the expected effective service time, the time that $\widetilde{\Omega}_{1_1}$'s state does not change due to services or outside factors.

     – The lower bound on the effective service time is $\frac{1}{\mu_1}$, corresponding to when neither node 1 nor node 2 are full. Therefore the upper bound on the effective service rate is μ_1.

     – The upper bound on the effective service time corresponds to the following cycle of events:

       * A customer, a, begins service at node 1, lasting on average $\frac{1}{\mu_1}$.

       * Customer a finishes service at node 1, but is blocked from transitioning to node 2, this lasts until the customer being served at

node 2, customer b finishes service. As all rates are Markovian then whichever point in b's service a gets blocked, a's expected wait is $\frac{1}{\mu_2}$.

∗ A customer, b, finishes service at node 2 and transitions to node 1. Now customer a moves to node 2, and another customer begins service at node 1, and so the cycle begins again.

This cycle is repeated with probability P_{repeat}. Therefore the upper bound for the effective service time is:

$$\begin{aligned}\frac{1}{m_1} &= \frac{1}{\mu_1} + \frac{1}{\mu_2} + P_{\text{repeat}}\left(\frac{1}{\mu_1} + \frac{1}{\mu_2} + P_{\text{repeat}}\left(\frac{1}{\mu_1} + \frac{1}{\mu_2} + P_{\text{repeat}}\Bigg(\cdots\right.\right.\\ &= \left(\frac{1}{\mu_1} + \frac{1}{\mu_2}\right) \times \left(1 + P_{\text{repeat}} + P_{\text{repeat}}^2 + P_{\text{repeat}}^3 + \dots\right)\\ &= \left(\frac{1}{\mu_1} + \frac{1}{\mu_2}\right) \times \left(\frac{1}{1 - P_{\text{repeat}}}\right)\end{aligned}$$

Let S_1 be the time a spends in service, and S_2 be the time b spends in service, then $S_1 \sim \text{Exp}(\mu_1)$ and $S_2 \sim \text{Exp}(\mu_2)$. Now $P_{\text{repeat}} = P(S_1 < S_2) = \frac{\mu_1}{\mu_1+\mu_2}$.

Therefore the lower bound on the effective service rate is:

$$\begin{aligned}
m_1 &= \frac{1}{\left(\frac{1}{\mu_1} + \frac{1}{\mu_2}\right)\left(\frac{1}{1-P_{\text{repeat}}}\right)} \\
&= \frac{1}{\left(\frac{1}{\mu_1} + \frac{1}{\mu_2}\right)\left(\frac{1}{1-\frac{\mu_1}{\mu_1+\mu_2}}\right)} \\
&= \frac{1}{\left(\frac{1}{\mu_1} + \frac{1}{\mu_2}\right)\left(\frac{\mu_1+\mu_2}{\mu_2}\right)} \\
&= \frac{1}{\frac{\mu_2(\mu_1+\mu_2)}{\mu_1\mu_2^2} + \frac{\mu_2(\mu_1+\mu_2)}{\mu_1\mu_2^2}} \\
&= \frac{\mu_1\mu_2^2}{(\mu_1+\mu_2)^2}
\end{aligned}$$

Now the actual effective service rate $\widetilde{\mu}_1$ for $\widetilde{\Omega}_{1_1}$ must lie in the interval (μ_1, m_1). Using Conjectures 1 and 2, we can conclude that $\widetilde{\omega}_{1_1} \leq \max(\omega^{\star}_{1_1}, \omega^{\star\star}_{1_1})$ when all other parameters are fixed.

- Consider the effective arrival rate of $\widetilde{\Omega}_{1_1}$. The effective arrival rate in $\widetilde{\Omega}_{1_1}$ is greater than or equal to the effective arrival rate in an Ω_{1_1} system; this is due to the extra customers who have transitioned from the node 2 to node 1. An increase in the arrival rate causes the mean time to deadlock to decrease.

Combining the above $\widetilde{\omega}_{1_1} \leq \omega_{1_1}$.

3. A similar argument yields $\widetilde{\omega}_{1_2} \leq \omega_{1_2}$.

Therefore

$$\begin{aligned}
\min(\widetilde{\omega}_{1_1}, \widetilde{\omega}_{1_2}, \widetilde{\omega}_2) &\leq \min(\omega_{1_1}, \omega_{1_2}, \omega_2) \\
\omega &\leq \min(\omega_{1_1}, \omega_{1_2}, \omega_2)
\end{aligned}$$

□

Λ_1	Λ_2	μ_1	μ_2	n_1	n_2	r_{11}	r_{12}	r_{21}	r_{22}	m_1	m_2	$\omega_{1_1}^{\star\star}$	$\omega_{1_1}^{\star}$	$\omega_{1_2}^{\star\star}$	$\omega_{1_2}^{\star}$	ω_2	$\min(\omega_{1_1}, \omega_{1_2}, \omega_2)$	ω
1.0	8.5	6.0	1.0	0	3	0.05	0.35	0.05	0.80	0.122	0.735	183.333	23.333	2.208	1.759	79.826	2.208	1.642
6.0	8.5	1.0	8.5	0	2	0.05	0.20	0.35	0.20	0.801	0.094	28.316	23.333	53.918	2.353	7.761	7.761	2.957
1.0	3.5	1.0	1.0	3	1	0.65	0.20	0.05	0.65	0.250	0.250	11.022	7.692	6.890	2.308	214.408	6.890	2.125
6.0	3.5	6.0	1.0	3	1	0.05	0.95	0.50	0.50	0.122	0.735	167.239	16.667	3.639	2.939	3.089	3.089	1.526
6.0	1.0	3.5	6.0	3	0	0.35	0.05	0.65	0.05	1.396	0.814	3.161	2.218	44.558	23.333	42.542	3.161	1.825
8.5	3.5	8.5	6.0	2	3	0.35	0.35	0.65	0.35	1.455	2.062	2.589	1.345	3.798	6.194	4.364	2.589	0.767
8.5	8.5	8.5	3.5	1	2	0.65	0.05	0.05	0.35	0.723	1.756	2.432	0.543	2.265	1.533	107.699	2.265	0.476
6.0	8.5	6.0	3.5	2	3	0.35	0.50	0.20	0.35	0.814	1.396	4.378	1.905	2.802	1.697	4.944	2.802	0.923
1.0	3.5	3.5	8.5	0	1	0.65	0.35	0.20	0.20	1.756	0.723	2.415	1.978	8.865	5.078	19.464	2.415	1.049
1.0	8.5	1.0	8.5	0	1	0.35	0.05	0.05	0.35	0.801	0.094	6.426	5.714	30.792	1.008	144.716	6.426	0.969
6.0	3.5	1.0	8.5	2	0	0.65	0.20	0.20	0.50	0.801	0.094	2.532	2.154	21.807	0.807	15.375	2.532	0.720
1.0	1.0	8.5	6.0	3	0	0.05	0.95	0.65	0.35	1.455	2.062	160.169	12103.030	4.243	3.333	27.803	4.243	1.163

Table 5.2: Table of example parameter sets illustrating the bound of Theorem 8.

Table 5.2 shows a few examples of this bound on some parameter sets. The first 10 columns show the parameters of the Ω system, the next seven columns show the intermediate derived parameters to use in the bound. The final column shows the actual time to deadlock of the Ω system.

This bound was tested on over 3 million parameter sets of the Ω system. In order to gain an understanding of how loose or tight this bound is, the following ratio is taken:

$$\beta = \frac{\omega}{\min(\omega_{1_1}, \omega_{1_2}, \omega_2)} \tag{5.63}$$

As both the numerator and denominator are positive, and $\omega \leq \min(\omega_{1_1}, \omega_{1_2}, \omega_2)$, then $0 \leq \beta \leq 1$. If β is close to 1 then the bound is tight, and if the β is close to 0 then there is a large percentage difference between the bound and the actual value for ω. Figure 5.36 shows a histogram of β for the parameter sets tested. Although the bound holds, it doesn't seem tight.

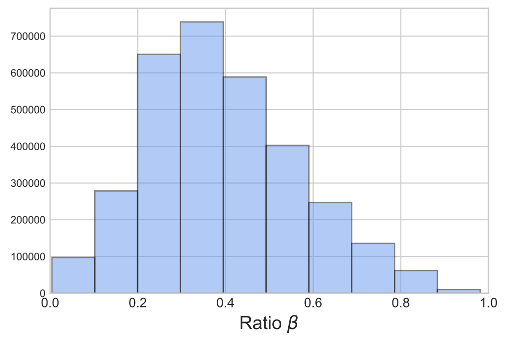


Figure 5.36: Normalised histogram of β, with estimated pdf.

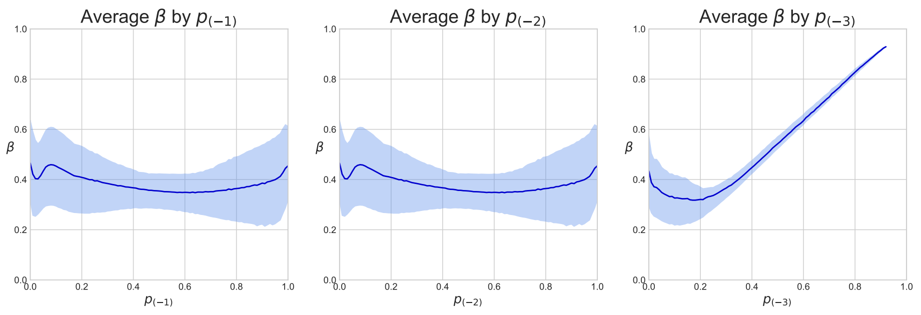


Figure 5.37: Average β by absorption probability.

The bound is closely associated with the expected times to the three deadlocks (-1), (-2) and (-3). The bound is tighter depending on which deadlock is most likely to occur. Recalling Section 5.1, let $p_{(-1)}$, $p_{(-2)}$ and $p_{(-3)}$ denote the probabilities of Ω reaching deadlocked state (-1), (-2) and (-3) respectively, calculated using Equation 5.2.

Figure 5.37 gives the average ratio β by each absorption probability $p_{(-1)}$, $p_{(-2)}$ and $p_{(-3)}$. The shaded areas show in interquartile range. Here it seems that deadlocks (-1) and (-2) have no effect on the tightness of the bound, however as the system is more likely to reach deadlock (-3) then the bound gets tighter. These plots however cannot tell the full story, especially at low values of $p_{(-1)}$, $p_{(-2)}$ and $p_{(-3)}$, as all three absorption probabilities are dependant on each other. For example, when $p_{(-3)}$ is small, this could either mean than $p_{(-1)}$ is large, $p_{(-2)}$ is large, both are moderate, or both are small. Each scenario produces a different time to deadlock, some are tight to the bound and others are not. This is shown by a large interquartile range for small $p_{(-3)}$. Similarly for the other absorption probabilities. In order to gain a fuller understanding, accounting for the absorption probabilities interacting with each other, Figure 5.38 shows a ternary plot [63] of the average ratio β against the absorption probabilities. It clearly shows the higher $p_{(-3)}$ the larger the ratio β,that is the tighter the bound. The probability of reaching deadlocked state (-1) or (-2) does not affect the tightness of the bound.

This highlights that the assumptions made about the $\widetilde{\Omega}_{1_1}$ and $\widetilde{\Omega}_{1_2}$ systems in the proof of Theorem 8 overestimate the difference between $\tilde{\omega}_{1_1}$ and ω_{1_1}, and between

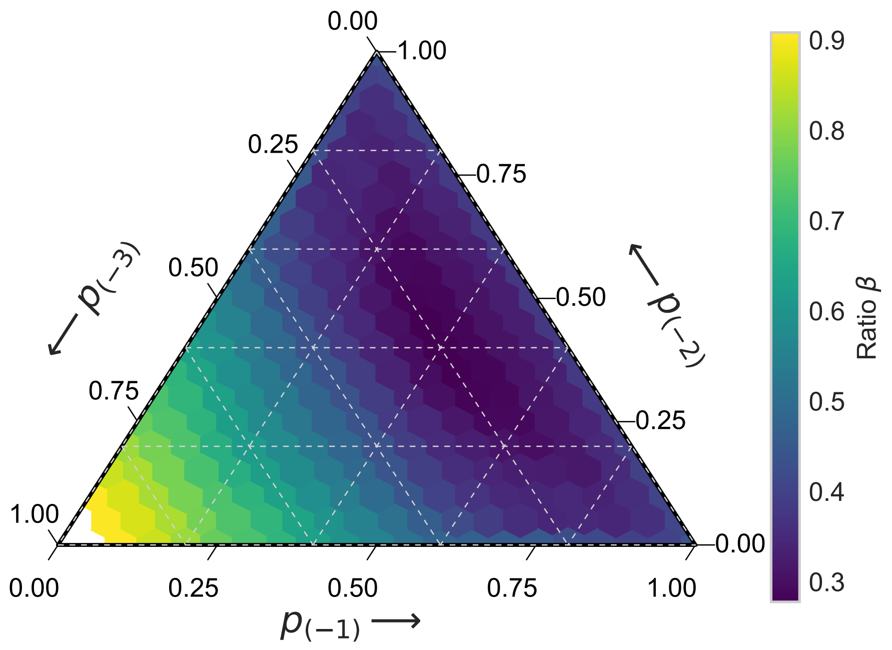


Figure 5.38: Ternary plot of absorption probabilities against β.

$\widetilde{\omega}_{1_2}$ and ω_{1_2}.

5.5 Summary

In this chapter Markov models of deadlocking queueing systems have been built. All two node queueing networks have been considered, except for the multi-server two node systems with self loops due to the size of the associated state space. For the one node system, further behaviour was modelled, namely customers baulking and scheduled server vacations.

These Markov models were used as tools to investigate the behaviour of the expected time to deadlock ω of the systems; in particular interest is taken in the effect of the system parameters on ω. Some intuitive results were found: the arrival rates and transition probabilities decrease the expected time to deadlock, while the queueing capacity and number of servers increase the time to deadlock. Particular interest was taken in the functions $\omega(\mu)$ and $\omega(\mu_1, \mu_2)$, as these are not monotonic. The effects of $\omega(\mu_1, \mu_2)$ were investigated in the limits of μ_1 and μ_2, with explanations proposed for the observed behaviours.

Finally, by considering single-server versions of networks (i) and (ii) as being embedded within the single-server version of network (iii), and considering the movements of customers, a bound on the time to deadlock of that system is found. A parameter sweeping analysis showed that this bound was tighter when the deadlock involving both nodes was most likely, but performed less well when deadlocks involving only one node was most likely.

All code used in the chapter is available at [125] with DOI `10.5281/zenodo.1420401`.

Chapter 6

Deadlock Resolution

This chapter investigates how deadlock is resolved. It is structured as follows:

- Section 6.1 introduces the concept of deadlock resolution.
- Section 6.2 formalises the language used to discuss sequences of unblockages.
- Section 6.3 introduces the concept of constrained unblockages, and presents results on how to choose constraints such that deadlock is guaranteed to be resolved.
- Section 6.4 shows that different deadlock resolution policies can lead to different non-deadlocked states. A Markov decision process is defined and used to find optimal deadlock resolution policies given some cost associated with a each deadlock resolution.
- Section 6.5 summarises the work of the chapter.

6.1 Introduction

Deadlock resolution procedures have been discussed in the literature. In Chapter 2 an overview of methods of dealing with deadlock was given. These were listed as deadlock prevention, deadlock avoidance, and deadlock detection and resolution, with prevention and avoidance deemed inappropriate for simulation models. For the deadlock detection procedures reviewed, only some discussed any subsequent deadlock resolution.

It is noted in [154] that “a deadlock requires the attention of a process outside those involved in the deadlock for its deletion and resolution”. That is, intuitively, that entities involved in the deadlock do not know they are in deadlock, and so cannot initiate a resolution procedure themselves. Some form of global decision maker, that same decision maker that detects the deadlock, must initiate the resolution procedure and carry out any associated decisions.

In general, resolution procedures abort or pre-empt a subset of the deadlocked jobs [97, 98, 105, 154]. Job abortion is described in [154]. This involves releasing the job’s held resources, cancelling the job’s resource requests, and starting the process afresh. For simulation modelling job abortion may be inappropriate, as processes could take twice as long, and some resource requests may never be successfully granted, deviating significantly from reality.

In [38] entities are either swapped or they seize resources, that is they displace other entities in order to resolve deadlocks. The aim of these actions is to break cycles in the closed strongly connected component of the entity / resource graphs, constructions that indicate deadlock. The displacement procedure is adapted in [176] in order to resolve deadlocks where entities request multiple resources before beginning an activity.

The choice of which deadlocked entity, or transaction, to pre-empt or abort is discussed in [105]. They list some deadlock resolution policies as: pre-empting any arbitrary transaction, pre-empting the transaction with the longest expected time to complete, pre-empting the transaction that would cost the least, and pre-empting the transaction which holds the least resources. The problem of optimally resolving deadlocks is addressed [97, 98]. Deadlock recovery here involves aborting a subset of jobs, where there is a cost associated with each aborted job, that minimises cost.

The problem of finding this subset is considered, and for most cases the problem is proved to be NP-complete.

The method that will be considered in this chapter is to create *virtual capacity* at a deadlocked node, and move a deadlocked customer into that introduced space. This causes a chain of unblockages that resolves the deadlock, and the virtual capacity is removed once it is no longer in use. A sequence of unblockages is required that ensures the virtual capacity is no longer required when the sequence terminates. All unblockages happen instantaneously without any simulation time passing, and so the rules of the system being simulated are only broken for an instant, and not for any meaningful amount of simulation time. The next section formalises discussion on sequences of unblockages in order to eventually prove the feasibility of this method if only a subset S of customers are allowed to be unblocked.

6.2 Sequential Unblockages

Often in queueing systems a blocked customer does not exist in isolation. That is, there may be another customer blocked to the node where the first customer is blocked. If the first customer is unblocked and moves to their destination node, then there is now space for the second customer to be unblocked. This chain of unblockages continues until some terminating criteria is reached. These unblockages, although occurring in sequence, happen instantly, with no time passing between the initial unblockage and the terminating criterion.

This section will now formalise the language used to represent sequences of unblockages. The following definition will be required:

Definition 9. *Define the directed multigraph $K = (V, E)$ which represents a deadlock of the queueing system. V corresponds to the set of nodes of the queueing system. E is the set of edges which represent blockage relationships in the following* *manner: there is an edge from $v_1 \in V$ to vertex $v_2 \in V$ if there is a customer blocked to the node corresponding to v_1 from the node corresponding to v_2.*

Recall from Chapter 2 that deadlock occurs if and only if the state digraph $D(t)$, from Definition 8, contains a knot. Note that:

- K is equivalent to taking the state digraph $D(t)$, contracting the vertices that correspond to servers in the same node (preserving multiple edges if they correspond to different blocked customers), and reversing the edges.
- K contains a knot subgraph, as contracting nodes does not affect connectivity, and reversing edges does not affect connectivity.

Consider that a sequence of unblockages is equivalent to the sequence $\{(E_n, v_n)\}$, where $E_n \subseteq E$ and $v_n \in V$. If at the n^{th} unblockage a customer is unblocked from the node corresponding to $u \in V$ to the node corresponding to $v_n \in V$, then:

$$(E_{n+1}, v_{n+1}) = (E_n \setminus \{(v_n, u)\}, u) \tag{6.1}$$

and $E_0 = E$, and $v_0 \in V$.

Now **the vertex v_n is interpreted as corresponding to a node which, at the n^{th} unblockage, has space for a customer to be unblocked to.** The successors of v_n correspond to the nodes which contain customers blocked to the node corresponding to v_n. Consider one such successor, u, whose corresponding node contains a customer to be unblocked to the node corresponding to v_n. u is arbitrary as there could be many customers blocked to the same node. Once that customer is unblocked, the node corresponding to v_n has their space filled, and the node corresponding to u has space to receive a blocked customer. Thus $v_{n+1} = u$, as given in Equation 6.1. As the edges of K correspond to blocked customers, the customer corresponding to the edge (v_n, u) is no longer blocked, and can be removed. Thus $E_{n+1} = E_n \setminus \{(v_n, u)\}$, as given in Equation 6.1.

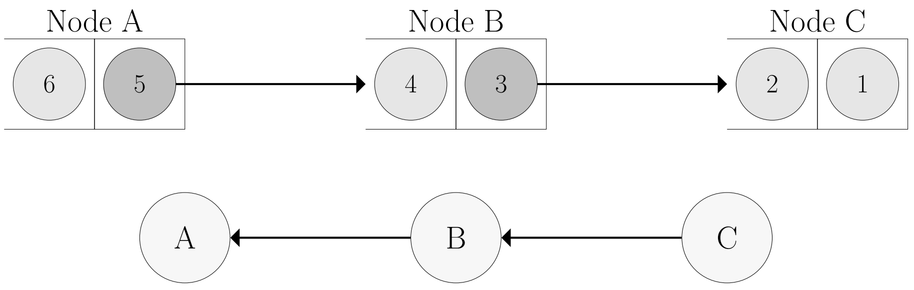


Figure 6.1: Three nodes with customers blocked in series, with corresponding $K = (V, E)$.

This sequence can be interpreted as traversing the edges of K, removing edges as they are traversed. Consider the blockages shown in Figure 6.1. This is a three node network, each with one server and space for one other customer to wait. Customer 5 at node A is blocked to node B, and customer 3 at node B is blocked to node C. The corresponding graph $K = (V, E)$ contains three vertices corresponding to nodes A, B and C, with edges $E = \{(C, B), (B, A)\}$.

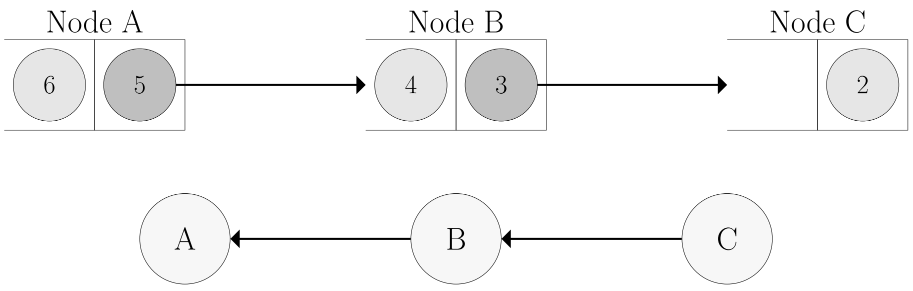
Now the customer being served at node C finishes service and leaves the system. This results in a chain of unblockages shown in Figure 6.2:

n = 0: Customer 1 finishes service and leaves node C. Customer 2 can begin service, and node C has space to potentially unblock other customers. Now $v_0 = C$, $E_0 = \{(C, B), (B, A)\}$. This is shown in Figure 6.2a.

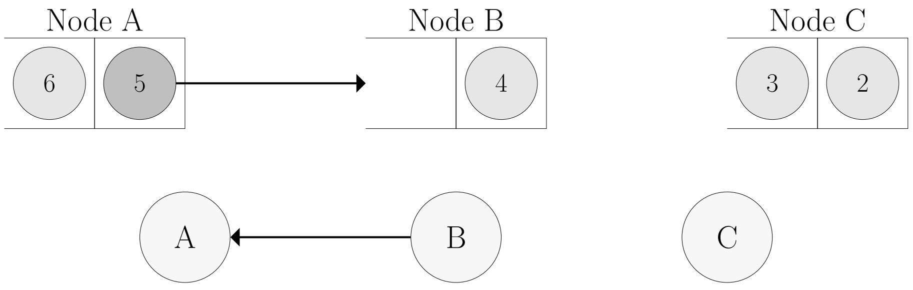
n = 1: Customer 3 is unblocked to node C. Customer 4 can begin service, and node B has space to potentially unblock other customers. Now $v_1 = B$, $E_1 = \{(B, A)\}$. This is shown in Figure 6.2b.

n = 2: Customer 5 is unblocked to node B. Customer 6 can begin service, and node A has space to potentially unblock other customers. Now $v_2 = A$, $E_2 = \{\}$. This is shown in Figure 6.2c.

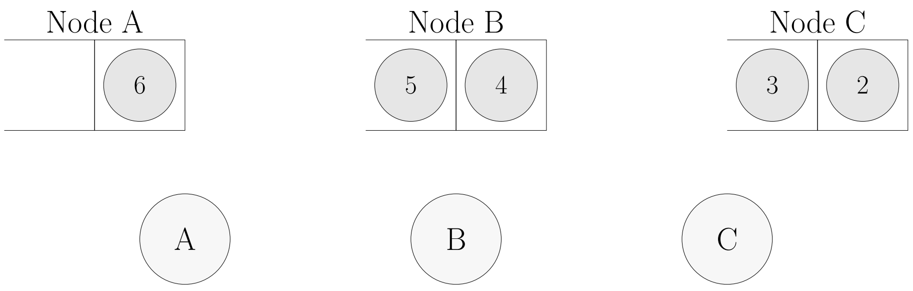
Consider a chain of unblockages that is initiated by moving a customer from the node corresponding to v_0 to some virtual capacity at the node corresponding to $v_\star$. It is not obvious when this sequence of unblockages ends. Theorem 9 gives terminating criteria, in terms of the sequence $\{(E_n, v_n)\}$, for the chain of unblockages.



(a) Unblockage step $n = 0$.



(b) Unblockage step $n = 1$.



(c) Unblockage step $n = 2$.

Figure 6.2: Illustrating the sequence (E_n, v_n) for a chain of unblockages.

Theorem 9. *Consider a queueing system and its corresponding graph $K = (V, E)$ as defined in Definition 9, and the sequence $\{(E_n, v_n)\}$ defined in Equation 6.1 on K. A sequence of unblockages, caused by moving a blocked customer to virtual capacity at the node corresponding to $v^\star \in V$, terminates at $\tilde{v}$ if and only if either:*

1. $deg^{out}(\tilde{v}) = 0$
2. $\tilde{v} = v^\star$

Note that this corresponds to introducing virtual capacity at a node, so that a customer can be moved there, creating capacity at its node, which in turn can be used to move a customer from an upstream node. This sequence of unblockages will terminate when there is no one else to move, or a customer is unblocked at the node where virtual capacity was originally introduced.

Proof. Note in the sequence of unblockages that virtual capacity is only introduced once, and removed as soon as it is no longer used. If the sequence does not terminate at $\tilde{v}$ then **both** of the following conditions must hold:

1. the node corresponding to $\tilde{v}$ has customers blocked to it. On K this corresponds to $\text{deg}^{\text{out}}(\tilde{v}) > 0$.
2. the node corresponding to $\tilde{v}$ has non-virtual capacity to potentially unblock another customer. On K only $v \in V \setminus \{v^\star\}$ have the potential for non-virtual capacity, thus $\tilde{v} \neq v^\star$.

A contradiction to both these statements being true is the statement in the theorem as required. That is either it does not terminate at $\tilde{v}$, or it terminates at $\tilde{v}$ because $\text{deg}^{\text{out}}(\tilde{v}) = 0$, or it terminates at $\tilde{v}$ because $\tilde{v} = v^\star$. □

6.3 Constrained unblockages

Define an S-constrained sequence of unblockages as follows:

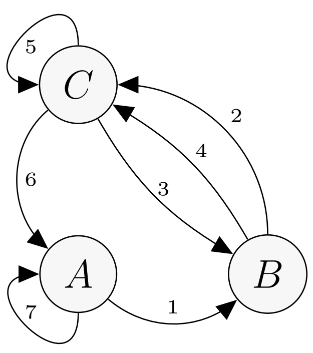
Definition 10. *Consider S a subset of blocked customers. A sequence of unblockages where the next customer to be unblocked is the customer from S who has been blocked the longest, is called S-constrained.*

Note that in an S-constrained sequence of unblockages, equivalent to the sequence $\{(E_n, v_n)\}$, at the n^{th} unblockage, the next customer to be unblocked is the one who has been waiting the longest to v_n. So if there is a choice of edges to traverse, there is no control over which edge to choose.

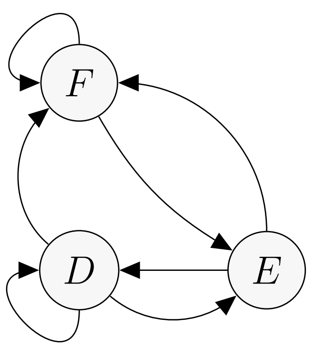
To resolve deadlock, it is required that there is an S-constrained sequence of unblockages which results in a system that is not in deadlock, and no node is using any virtual capacity. This section proves that for the deadlock resolution method discussed above, that is moving a blocked customer to the back of its destination node, despite the lack of capacity, S can be chosen such that there is an S-constrained sequence of unblockages that fulfils the requirement.

Consider $K = (V, E)$ given by Definition 9 for a general deadlock. The following notation will be used throughout this section:

- S is the chosen set of customers that are allowed to be unblocked in an S-constrained sequence of unblockages.
- V is the set of vertices corresponding to nodes of the queueing network.
- E is the set of edges corresponding to *all* blocked customers of the system.
- $\hat{E} \subseteq E$ is the set of edges corresponding to *deadlocked* customers in the system.
- $\tilde{E} \subseteq E$ is the set of edges corresponding to the customers in S.
- $\hat{K} = (V, \hat{E})$ and $\tilde{K} = (V, \tilde{E})$ are edge-induced subgraphs of K.



(a) Eulerian



(b) Not Eulerian.

Figure 6.3: Examples of digraphs which are a) Eulerian, and b) not Eulerian.

- $\{E_n\}$ is the sequence of edge sets described in Equation 6.1.
- $\{v_n\}$ is the sequence of vertices described in Equation 6.1.

Before stating the next theorem, recall that a directed graph is Eulerian if there is a circuit that traverses every edge of the graph [187]. Also a theorem in [64] gives that a directed graph is Eulerian if all vertices have equal in and out degree. This ensures that it is always possible for a trail to reach its origin regardless of the route taken. Examples of multi-digraphs which are Eulerian and not Eulerian are given in Figure 6.3. For the Eulerian digraph, edges have been numbered to give one example of an Eulerian circuit.

Notice the degrees of each vertex: for the Eulerian digraph in Figure 6.3a:

- $\deg^{\text{out}}(A) = 2 = \deg^{\text{in}}(A)$
- $\deg^{\text{out}}(B) = 2 = \deg^{\text{in}}(B)$
- $\deg^{\text{out}}(C) = 3 = \deg^{\text{in}}(C)$

and for the digraph which is not Eulerian in Figure 6.3b:

- $\deg^{\text{out}}(D) = 3 \neq 2 = \deg^{\text{in}}(D)$
- $\deg^{\text{out}}(E) = 2 = \deg^{\text{in}}(E)$
- $\deg^{\text{out}}(F) = 2 \neq 3 = \deg^{\text{in}}(F)$

Now Theorem 10 states that by only allowing unblockages from a certain subset of blocked customers, then the required conditions for feasible deadlock resolution are satisfied.

Theorem 10. *Consider a situation where deadlock has occurred. A blocked customer at the node corresponding to $v_0 \in V$ is moved to virtual capacity at the queue at the node corresponding to $v^\star \in V$. Given a subset of blocked customers S, and corresponding edges $\tilde{E}$, the system will be resolved of deadlock, and no node is using virtual capacitxsy, after an S-constrained sequence of unblockages, if both:*

1. *$\tilde{E} \subseteq \hat{E}$; and*
2. *$\tilde{K} = (V, \tilde{E})$ is Eulerian.*

Proof. The S-constrained sequence of unblockages is equivalent to traversing the edges of $\tilde{K} = (V, \tilde{E})$, beginning at v_0, removing edges as they are traversed, or the sequence $\{(E_n, v_n)\}$ where $E_0 = \tilde{E}$. Theorem 9 gives that this sequence terminates at $\tilde{v}$ if $\tilde{v} = v_\star$, where the virtual capacity at $v^\star$ is removed; **or** if $\deg^{\text{out}}(\tilde{v}) = 0$ where there is no customer blocked to the node corresponding to v_n. S should be chosen such that the only terminating criterion is $\tilde{v} = v_\star$.

Assume the sequence terminates because $\deg^{\text{out}}(\tilde{v}) = 0$, then **either**:

1. the $\tilde{v} \in V$ corresponds to a node not visited in the sequence before, that had no customers blocked to it. Thus that node was not deadlocked, and so on K, $\hat{E} \subset \tilde{E}$.
2. the node corresponding to $\tilde{v}$ was deadlocked, however the final in-edge of that node was traversed and there are no out-edges. As $v^\star$ has not been reached, K cannot be Eulerian.

A contradiction to both those statements being true is the statement in the theorem as required. □

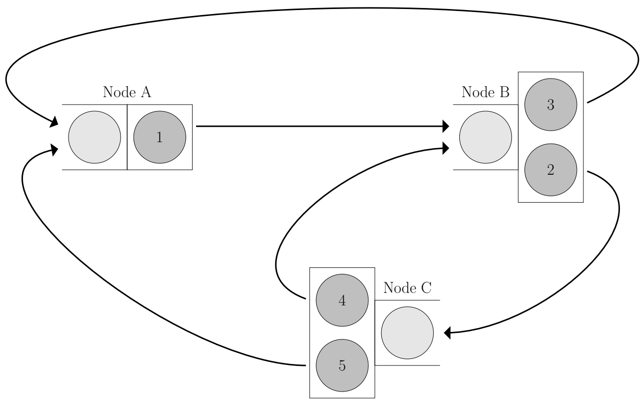


Figure 6.4: Example of a deadlock where constrained unblockages are required.

An immediate consequence of this theorem is that at any deadlock, a subset of blocked customers S can be chosen that will be eligible for unblockages such that deadlock resolution does not break any rules of the queueing system. Note that this is not a constructive result.

An example of a system where constraining S is required to remove virtual capacity when resolving deadlock is given. Consider the three node system shown in Figure 6.4, where node A has one server, and nodes B and C have two servers. Customer 1 at node A is blocked to node B; customer 3 at node B is blocked to node A; customer 2 at node B is blocked to node C; customer 4 at node C is blocked to node B; and customer 5 at node C is blocked to node A.

The corresponding graph K for this system is given in Figure 6.5. Note that K is not Eulerian. Allowing a sequence of unblockages without constraining the set of blocked customers that are allowed to be unblocked may not result in the virtual capacity being removed.

Consider the case where virtual capacity is introduced at node A, and customer 5 at node C is moved there. Now $v_\star = A$ and:

n = 0: Node C has space to potentially unblock other customers, and $v_0 = C$, $E_0 = \{(B, A), (C, B), (A, B), (B, C)\}$.

n = 1: Customer 2 is unblocked to node C, node B has space to potentially unblock

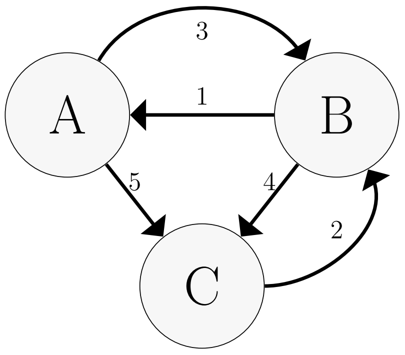


Figure 6.5: Graph K, for the example in Figure 6.5.

other customers and $v_1 = B$, $E_1 = \{(B, A), (A, B), (B, C)\}$.

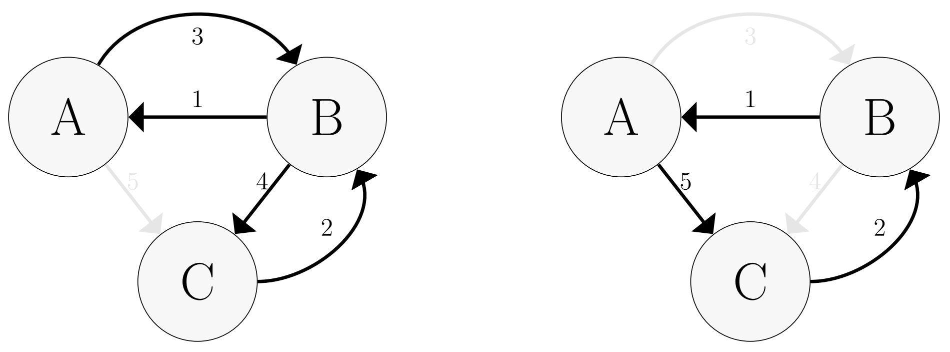
n = 2: Customer 4 in unblocked to node B, node C has space to potentially unblock other customers, and $v_2 = C$, $E_2 = \{(B, A), (A, B)\}$.

There are no edges out of C in E_2, therefore no more unblockages can occur. However there is still virtual capacity being occupied at node A. The deadlock could not be adequately resolved without constraining the set of customers who are allowed to be unblocked.

From Theorem 10 a subset of blocked customers S can be chosen such that no rules of the queueing system are broken. For the above system, two possible subsets of customers to allow to be unblocked would be: $S_1 = \{1, 2, 3, 4\}$, that is disallowing customer 5 from being unblocked, removing edge (A, C) from K; and $S_2 = \{1, 2, 5\}$, disallowing customers 3 and 4 from being unblocked, removing edges (A, B) and (B, C) from K. The corresponding $\tilde{K}$ for these possibilities are shown in Figure 6.6. Note that both are Eulerian.

Finding the largest subset of customers to allow to be unblocked is equivalent to the "Directed Eulerian Edge Deletion" problem for the graph $\hat{K} = (V, \hat{E})$. This problem is NP-hard and solvable in exponential time [26, 34, 58]. The following result gives a trivial constructive argument for always being able to resolve deadlock.

Theorem 11. *Consider a situation where deadlock has occurred. Choosing S to correspond to any cycle in $\hat{K} = (V, \hat{E})$ is sufficient to resolve deadlock.*



(a) Removing edge (A, C). (b) Removing edges (A, B) and (B, C).

Figure 6.6: Two possibilities for making K Eulerian.

Proof. Recall that $\hat{K}$ is a knot, and so a cycle, which is Eulerian, exists. The result is ensured by Theorem 10. $\square$

Note that Theorem 11 gives a sufficient approach to ensure that deadlock can always be resolved. This requires constraining the deadlocked customers, and so sometimes customers who have been waiting longer than others will still be waiting until deadlock is fully resolved. The following results will describe specific queueing systems where no such constraint is required.

Theorem 12. *At deadlock, for queueing systems where each node has only one server, choosing S such that $\tilde{E} = \hat{E}$ is sufficient to ensure that any virtual capacity is removed.*

Proof. Consider the set of nodes involved in the deadlock. Then each of those nodes has one and only one blocked customer. Thus each corresponding vertex in V has exactly one in-edge, which creates a cycle. From Theorem 11 the result follows. $\square$

Theorem 13. *At deadlock, for two node queueing systems, choosing S such that $\tilde{E} = \hat{E}$ is sufficient to ensure that any virtual capacity is removed.*

Proof. There are two cases to consider:

- Consider than only one node is involved in the deadlock; then every customer at that node is blocked to itself, and so every out-edge of the corresponding vertex connects to itself, thus $\hat{K}$ is Eulerian.

- Consider that both nodes, A and B, are involved in the deadlock. Then there is at least one customer blocked from node A to node B, and at least one customer blocked from node B to node A, so $\hat{K}$ contains a cycle. As there are only two vertices and $\hat{K}$ contains a cycle, while traversing the edges, every vertex will be visited before traversing the final out-edge of any vertex. Thus any node is reachable from any other node regardless of which route is taken.

□

In the next section these theorems will be used to investigate different approaches to resolving deadlock, and their effect on the system.

6.4 Resolution policies

In all but the most trivial system, the one node single server system, there is a choice of customers to move when resolving deadlock. In many cases this choice may lead to different resulting non-deadlocked states. The order in which customers were blocked leading up to the deadlock is vital in determining the correct sequence of unblockages.

Consider the deadlock shown in Figure 6.7. Here there are two nodes, both are involved in the deadlock. Node A has two servers and node B has a single server. Customers 1 and 5 are blocked to node A, and customer 2 is blocked to node B. The customers were blocked in the following order: 1, 5 and then 2.

There are three possible actions to take to resolve this deadlock: move customer 1 to virtual capacity at node A, move customer 2 to virtual capacity at node B, and move customer 5 to virtual capacity at node A. As this is a two node system, Theorem 13 gives that allowing all deadlocked customers, in this case all blocked customers, to be unblocked resolved the deadlock adequately.

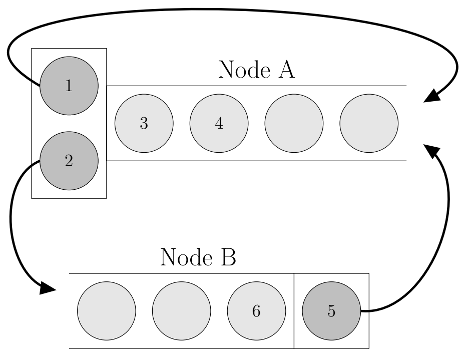
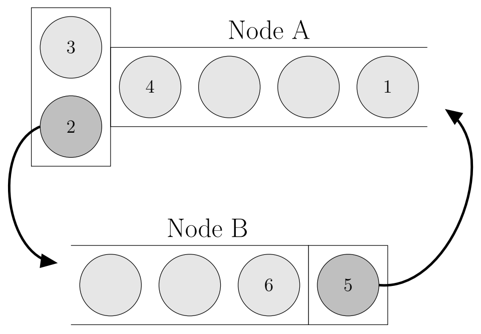


Figure 6.7: Two node queueing network in deadlock.

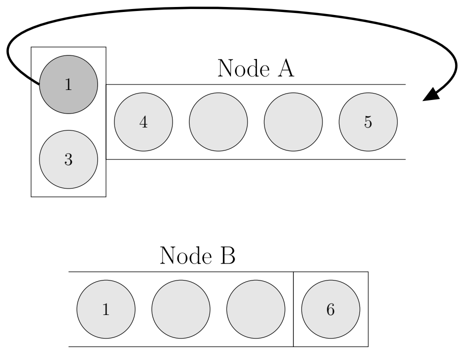
- Consider the case where customer 1 is moved to virtual capacity at node A. This allows customer 3 to begin service, all customers in the queue move, and customer 1 now occupies non-virtual capacity at node A. Deadlock is resolved as not all servers contain a blocked individual, however two customers are still blocked, customers 2 and 5. This final non-deadlocked state is shown in Figure 6.8a.

- Consider the case where customer 5 is moved to virtual capacity at node A. This allows customer 6 to begin service, all customers in the queue move, and there is an vacant space at the back of queue B. Now customer 2 is unblocked, moving to that space. This allows customer 3 to begin service, all customers in the queue move, and customer 5 now occupies non-virtual capacity at node A. Deadlock is resolved as not all servers contain a blocked individual, however customer 1 is still blocked. This final non-deadlocked state is shown in Figure 6.8b.

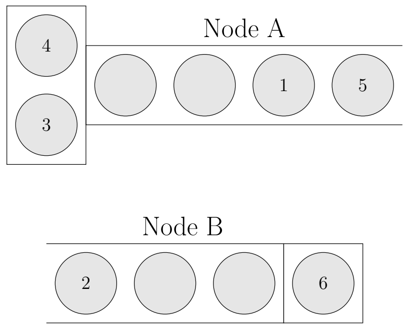
- Consider the case where customer 2 is moved to virtual capacity at node B. This allows customer 3 to begin service, all customers in the queue move, and there is an vacant space at the back of queue A. Now customer 1 is unblocked, moving to that space. This allows customer 4 to begin service, all customers in the queue move, and again there is an vacant space at the back of queue A.



(a) Move customer 1.



(b) Move customer at 5.



(c) Move customer 2.

Figure 6.8: Three different resulting non-deadlocked states.

> Now customer 5 in unblocked, moving to that space. This allows customer 6 to begin service, all customers in the queue move, and customer 2 now occupies non-virtual capacity at node B. Deadlock is resolved as there are no blocked individuals. This final non-deadlocked state is shown in Figure 6.8c.

Therefore the choice of action can have an effect on the outcome of the deadlock resolution. For multi-server queueing systems with more than two nodes, there are further choices, namely which subset of customers S to allow to be unblocked?

To further illustrate and explore these concepts, the sub-case of two node systems will only be considered.

It is reasonable to assume that in systems that may reach deadlock, there is a cost associated with its resolution. In computer and communications systems this may be a computational cost, corresponding to a lack of performance such as manually breaking the flow of the system, and to running edge deletion algorithms. In human systems, such as road traffic and healthcare systems, deadlock resolutions may be carried out by traffic control vehicles or by utilising extra stretchers to move patients. These actions increase workload and may exhibit other equipment or time costs.

Given this, can the system be controlled such that costs are minimised? This section considers finding optimal deadlock resolution policies, that is which individual to put in virtual capacity, to minimise this speculated cost. As a proof of concept, a Markov decision process (MDP) for two arbitrary illustrative examples are defined, and solved using dynamic programming. Generalisations are discussed but are left as further work.

6.4.1 Markov decision processes

A general MDP [138] has states, actions, transition probabilities and rewards. For the optimal deadlock resolution problem, the MDP takes the form:

$$\{Z, A_z, p(z_1, z_2, a), \omega(z_1, z_2, a)\}$$

where:

- Z is the set of deadlocked states (To be defined in Section 6.4.3).
- A_z is the set of actions that can be performed when in state $z \in Z$, that is which deadlocked customer to place in virtual capacity.
- $p(z_1, z_2, a)$ are the probabilities of reaching state $z_2 \in Z$ from state $z_1 \in Z$ given that action $a \in A_{z_1}$ was taken.
- $\omega(z_1, z_2, a)$ is the probabilistic reward received for reaching state $z_2 \in Z$ after taking action $a \in A_{z_1}$ when in state $z_1 \in Z$. This corresponds to the inter-deadlock time, that is the time to deadlock state z_2 from the state immediately after taking resolution action a when in deadlock state z_1.

For this deadlock resolution problem, the decision epochs are discrete, and correspond to the times at which the system reaches a deadlocked state. Although these decision epochs happen in random continuous time, this is accounted for in the rewards $\omega(z_1, z_2, a)$, therefore real time can be ignored, and this problem can be thought of in discrete terms.

The set of actions A_z for each $z \in Z$ corresponds to which customer is to be moved. The elements $a \in A_z$ take the form (n, d), meaning that a customer at node n who is blocked to node d should be moved to virtual capacity at node d. Not every action is feasible: in order to take action (n, d) there must be a customer at node n that is blocked to node d; and that customer must be in the constrained set of customers S. That is the edge $(n, d) \in \tilde{E}$.

Consider in an infinite simulation run, a fixed, positive cost is expended each time deadlock is reached and resolved. Minimising the total cost is equivalent to maximising the inter-deadlock times. So the aim is to find the optimal policy $\pi(z) \in A_z$

for all $z \in Z$, such that taking action $\pi(z)$ in state z maximises the inter deadlock times $\omega(z_1, z_2, a)$, or the rewards, of a simulation run.

A dynamic programming method used to obtain the optimal policy is described in the next section. Here the expected value $\mathbb{E}\left[\omega(z_1, z_2, a)\right]$ is used in place of the probabilistic rewards. As there are not yet any closed from for $p(z_1, z_2, a)$ and $\mathbb{E}\left[\omega(z_1, z_2, a)\right]$, simulations are run to find empirical values.

6.4.2 Solution method

MDPs may be solved by using dynamic programming and reinforcement learning algorithms. In this section one particular dynamic programming method, value iteration, is outlined.

Dynamic programming algorithms solve MDPs by maximising the total expected reward. For infinite horizon problems, that is problems that do not have a specified termination, this becomes unbounded. In order to account for this, a discount factor, or discount rate γ ($0 \leq \gamma < 1$) is introduced [138]. This ensures that a future reward has less value than present reward.

A policy calculated using a high discount rate will always strive to maximise future rewards. This policy may choose actions that won't immediately return the highest rewards, but will allow the agent to reach states that may give even higher rewards in the future; or avoid actions that immediately give high rewards if they allow the agent to enter states in the future which only yield low rewards. A policy calculated using a low discount rate will strive for immediate rewards, and it's future outlook is short. If $\gamma = 0$ then the agent is myopic and so is only concerned with maximising its immediate rewards, with no thought about which states, possible action sets, and rewards may arise in the future as a consequence of this.

Value iteration is an iterative algorithm, that updates $V(z)$ values. These values represent the expected future reward pay-off from state $z \in Z$, given that a policy is followed. It has been shown that values converge to $V^*(z)$, the expected future reward pay-off from state $z \in Z$, given that an optimal policy is followed. Optimal policies may be found once the optimal value function is found, that which satisfies the Bellman optimality equation [158] defined for this problem in Equation 6.2.

$$V^*(z) = \max_a \sum_{z'} p(z', z, a) \left[\mathbb{E}\left[\omega(z, z', a)\right] + \gamma V^*(z')\right] \tag{6.2}$$

Algorithm 4 shows the Value iteration algorithm [158]. The algorithm runs until be values come within ϵ of convergence.

Algorithm 4: The value iteration algorithm [158].

```
$V(z) = 0$ for all $z \in Z$
repeat
    $\Delta \longleftarrow 0$
    for $z \in Z$ do
        $v \longleftarrow V(z)$
        $V(z) \longleftarrow \max_a \sum_{z'} p(z', z, a) \left[\mathbb{E}\left[\omega(z, z', a)\right] + \gamma V(z')\right]$
        $\Delta \longleftarrow \max(\Delta, |v - V(z)|)$
    end
until $\Delta < \epsilon$
```

Output: $\pi(z) = \arg\max_a \sum_{z'} p(z', z, a) \left[\mathbb{E}\left[\omega(z, z', a)\right] + \gamma V(z')\right]$ for all $z \in Z$

In order to compare the effect of short, medium and high discount rates, the algorithm is run using $\gamma = 0.1$, $\gamma = 0.5$ and $\gamma = 0.9$. A value of $\epsilon = 10^{-7}$ is used for the termination criterion.

6.4.3 The state space

The state space for this Markov decision process is the set of all deadlock states of the system. A new system state for a queueing network is now defined. The states defined here allow differentiation between seemingly identical configurations of blocked customers, but are however different in the destinations to which customers are blocked to, and the order in which they were blocked. A deadlock state is a unique configuration of blocked customers who are experiencing deadlock; a configuration means the combination of which customers blocked at certain servers to certain nodes, and also the order of blockages. Note that the deadlock configurations defined here are starkly different from those discussed in Chapter 2.

In order to keep track of the blockage destinations of customers, as well as the order of blockages, a tuple of arrays, $(\underline{B}, \underline{\kappa})$, is used. For a system with N nodes, $\underline{\kappa}$ is a one dimensional array of length N denoting the number of customers at each node; that is the ith entry denotes the number of customers waiting, in service or blocked at the ith node. $\underline{B}$ is a two dimensional array of size $N \times N$, whose entries are themselves arrays of varying sizes; the ijth entry shows the overall order in which customers were blocked from the ith node to the jth node.

An example is shown in Equation 6.3, where $\emptyset$ denotes an array of no length.

$$\underline{B} = \begin{pmatrix} (2,3) & \emptyset & \emptyset \\ \emptyset & \emptyset & (1,5) \\ \emptyset & (4) & \emptyset \end{pmatrix} \qquad \underline{\kappa} = \begin{pmatrix} 5 & 5 & 2 \end{pmatrix} \tag{6.3}$$

This denotes a state in a three node network. The second part of the tuple, $\underline{\kappa} = (5, 5, 2)$ indicates that there are 5 customers present at the first node, 5 customers present at the second node, and 2 customers present at the third. The first part of the tuple, $\underline{B}$, shows the destination to which customers are blocked, and the order in which they became blocked. In this case, of the blocked customers:

- The first customer blocked was from node 2 to node 3, as $1 \in \underline{B}_{2,3}$,
- the second customer blocked was from node 1 to node 1, as $2 \in \underline{B}_{1,1}$,
- the third customer blocked was from node 1 to node 1, as $3 \in \underline{B}_{1,1}$,
- the fourth customer blocked was from node 3 to node 2, as $4 \in \underline{B}_{3,2}$,

- and the fifth customer blocked was from node 2 to node 3, as $5 \in \underline{B}_{2,3}$.

For the MDP of interest, the state space Z contains deadlocked states only, defined in the above manner. Finding all possible deadlocked states of a given system is challenging, and for the illustrative examples in this chapter they were found manually.

6.4.4 Policy performance

In order to compare the policies obtained using the dynamic programming method above, the following evaluation method is proposed. A Ciw simulation model will be run for a finite amount of time T, resolving any deadlock encountered according to the evaluated policy. Assume T large enough such that the simulation run does not terminate before a sufficient amount of deadlock resolutions has taken place. Denote the ith reward received by r_i, and say there were Ξ_T deadlock resolutions during the run time T. Recall that successive rewards r_i are the inter-deadlock times, then the total reward received is:

$$\sum_{i=0}^{\Xi_T} r_i = T - \omega_0 - e \tag{6.4}$$

where ω_0 is the time to the first deadlock from an empty systems, and e is the residual time from the last encountered deadlock to the end of the simulation run. Due to the memoryless property of Markovian systems, e, ω_0 and T are independent of the policy to be evaluated. Therefore taking the total reward received as a performance measure will not evaluate the policy.

An alternative performance measure would be to take the random variable Ξ_T, the number of deadlock resolutions in time T. This has a closer meaning to the original problem, which is to minimise the total deadlock resolution costs, which was fixed and incurred each time a deadlock resolution took place. Then the total cost in time T would be Ξ_T multiplied by this cost. Therefore the number of deadlock resolutions in time T, Ξ_T, is taken as a measure of policy performance.

In order to evaluate the performance of any derived policy, the values Ξ_T for the derived policy are compared with the corresponding statistics for a random policy. That is a policy that uniformly randomly chooses an action to take from all feasible action choices.

6.4.5 Two proof-of-concept examples

This section will introduce two illustrative examples of queueing networks and their optimal control of deadlock resolution policies are found using dynamic programming. Here their MDP formulations will be given, and solutions using value iteration will be presented.

As mentioned previously, as of yet there are no closed form expressions for the probabilities $p(z_1, z_2, a)$ or the rewards $\omega(z_1, z_2, a)$. Empirical values for these are found by running simulations with Ciw. A simulation was run 10,000 times for each state-action pair. A run of the simulation involves setting up the simulation and artificially inserting and blocking individuals until the desired starting state $z_1 \in Z$ was reached. Then an action $a \in A_{z_1}$ was taken to resolve the deadlock, and the simulation was left to run until another deadlock was reached once more. The resulting deadlock state $z_2 \in Z$ was recorded along with the inter-deadlock time $\omega(z_1, z_2, a)$.

Probability density functions $p(z_1, z_2, a)$ are calculated using:

$$p(z_1, z_2, a) = \frac{n}{10000} \tag{6.5}$$

where n is the number of times state z_2 was reached from state z_1 when taking action a. Average inter-deadlock times are used as rewards, and these measured values are then used in the value iteration algorithm described in Section 6.4.2.

Illustrative example I

Consider the deadlocking queueing system shown in Figure 6.9. The parameters of this system are summarised in the Ciw parameters shown in Figure 6.10.

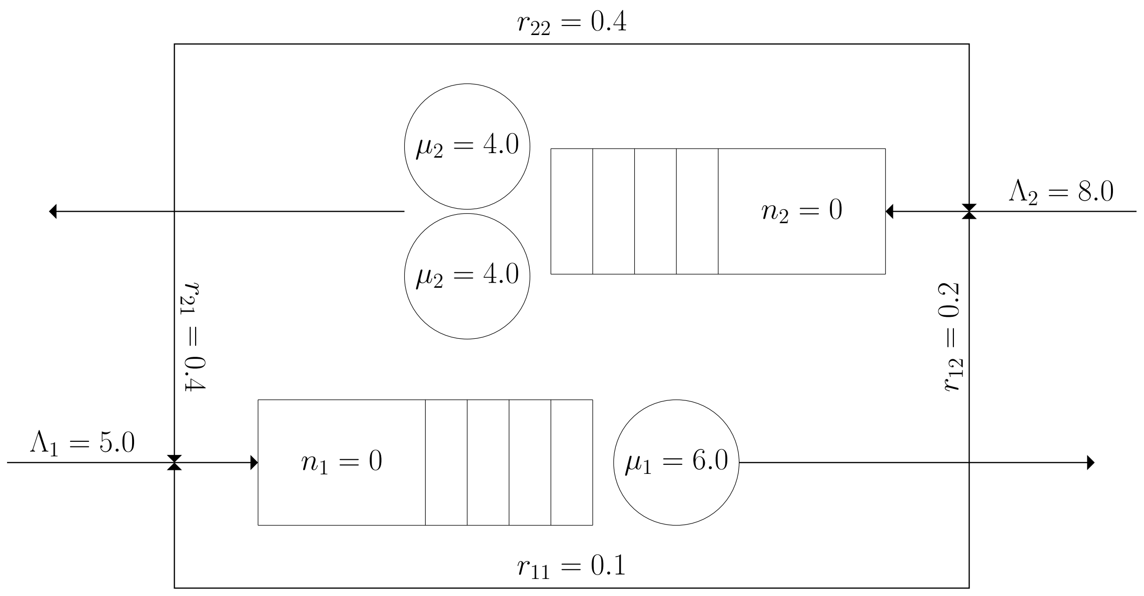


Figure 6.9: Diagram of the queueing network for illustrative example I.

```
>>> N = create_network(
...     Arrival_distributions=[['Exponential', 5.0],
...                            ['Exponential', 8.0]],
...     Service_distributions=[['Exponential', 6.0],
...                            ['Exponential', 4.0]],
...     Transition_matrices=[[0.1, 0.2],
...                          [0.4, 0.4]],
...     Number_of_servers=[1, 2],
...     Queue_capacities=[0, 0]
... )
```

Figure 6.10: Ciw parameters for illustrative example I.

This system has 22 deadlock states. These are listed in Appendix B. Heat-maps illustrating the probabilities $p(z_1, z_2, a)$ and the rewards $\omega(z_1, z_2, a)$ are also given in that appendix.

The set of all actions is $A = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. Not all actions are feasible, so $A_s \subseteq A$ for all $s \in S$. Table 6.1 summarises the feasible actions for each state.

Action	Feasible action sets
(1, 1)	$A_{z_{01}}, \dots, A_{z_{09}}$
(1, 2)	$A_{z_{14}}, \dots, A_{z_{22}}$
(2, 1)	$A_{z_{14}}, \dots, A_{z_{22}}$
(2, 2)	$A_{z_{10}}, \dots, A_{z_{13}}, A_{z_{17}}, \dots, A_{z_{22}}$

Table 6.1: Summary of which feasible actions sets A_z contain each action $a \in A$ for illustrative example I.

Illustrative example II

Consider the deadlocking queueing system shown in Figure 6.11. The parameters of this system are summarised in the Ciw parameters shown in Figure 6.12.

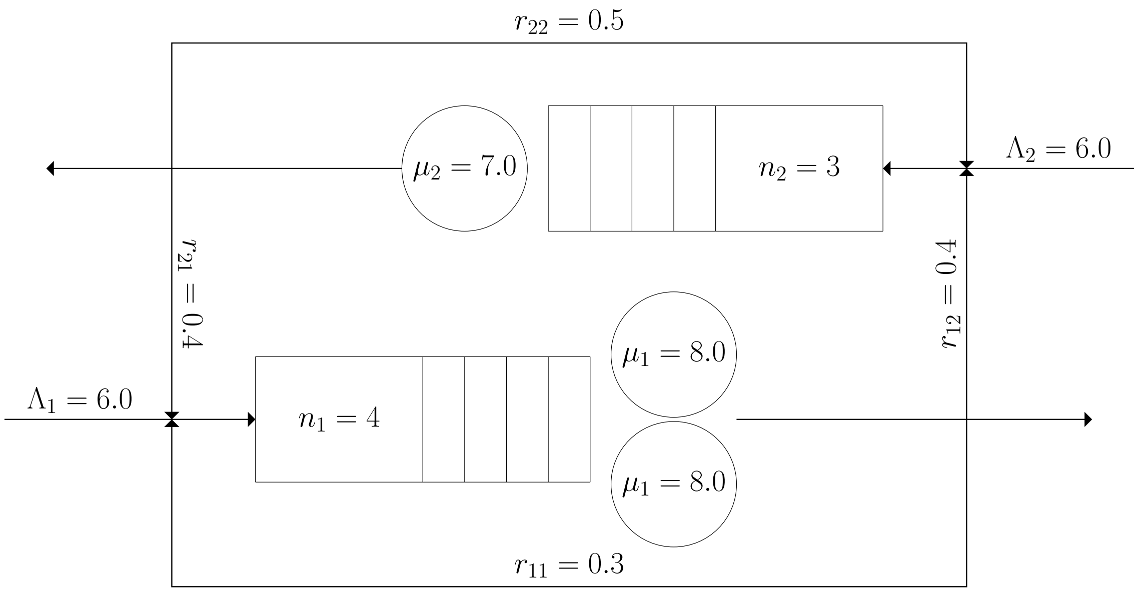


Figure 6.11: Diagram of the queueing network for illustrative example II.

```
>>> N = create_network(
...     Arrival_distributions=[['Exponential', 6.0],
...                            ['Exponential', 6.0]],
...     Service_distributions=[['Exponential', 8.0],
...                            ['Exponential', 7.0]],
...     Transition_matrices=[[0.3, 0.4],
...                          [0.4, 0.5]],
...     Number_of_servers=[2, 1],
...     Queue_capacities=[4, 3]
... )
```

Figure 6.12: Ciw parameters for illustrative example II.

This system has 43 deadlock states. These are listed in Appendix C. Heat-maps

Action	Feasible action sets
(1, 1)	$A_{z_{22}}, \dots, A_{z_{34}}, A_{z_{38}}, \dots, A_{z_{43}}$
(1, 2)	$A_{z_{35}}, \dots, A_{z_{43}}$
(2, 1)	$A_{z_{35}}, \dots, A_{z_{43}}$
(2, 2)	$A_{z_{01}}, \dots, A_{z_{21}}$

Table 6.2: Summary of which feasible actions sets A_z contain each action $a \in A$ for illustrative example II.

illustrating the probabilities $p(z_1, z_2, a)$ and the rewards $\omega(z_1, z_2, a)$ are also given in that appendix.

The set of all actions is $A = \{(1,1), (1,2), (2,1), (2,2)\}$. Not all actions are feasible, so $A_s \subseteq A$ for all $s \in S$. Table 6.2 summarises the feasible actions for each state.

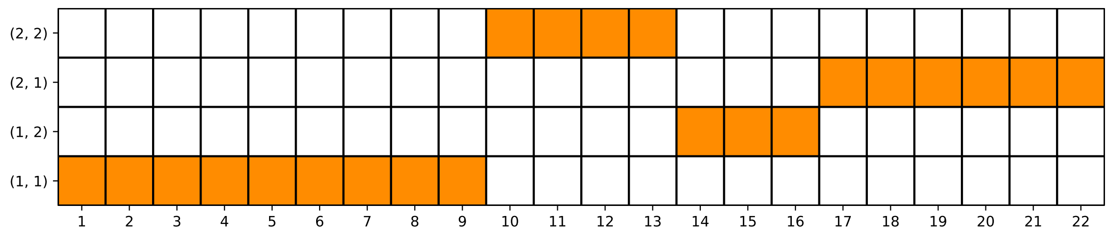
Solution & evaluation

For both examples a low discount rate ($\gamma = 0.1$) yielded a different optimal policy to a high discount rate ($\gamma = 0.5$ and $\gamma = 0.9$). In both examples the two policies differ by three states.

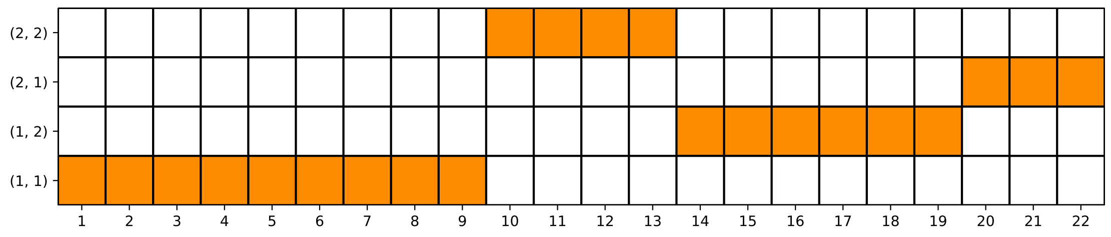
Figures 6.13 and 6.14 give visual representations for the derived optimal policies (low and high discount rates) for illustrative examples I and II respectively. Orange squares indicate that that state-action pair belongs in the policy.

The optimal derived policies for both illustrative example were evaluated against the random policy in each case. Simulations were run for $T = 50$ time units. Figures 6.15a and 6.15b give the distribution of the performance measure Ξ_{50} for the optimal and random policies. For both examples the measures of centrality of Ξ_{50} for the optimal policies are slightly lower than the random policy, however this improvement is very small in comparison to using the random policy.

Independent sample t-tests were carried out in order to assert whether the small improvements seen in Figures 6.15a and 6.15b are significant, or are explained by random sampling. The null hypotheses are that the mean Ξ_{50} value for the ran-

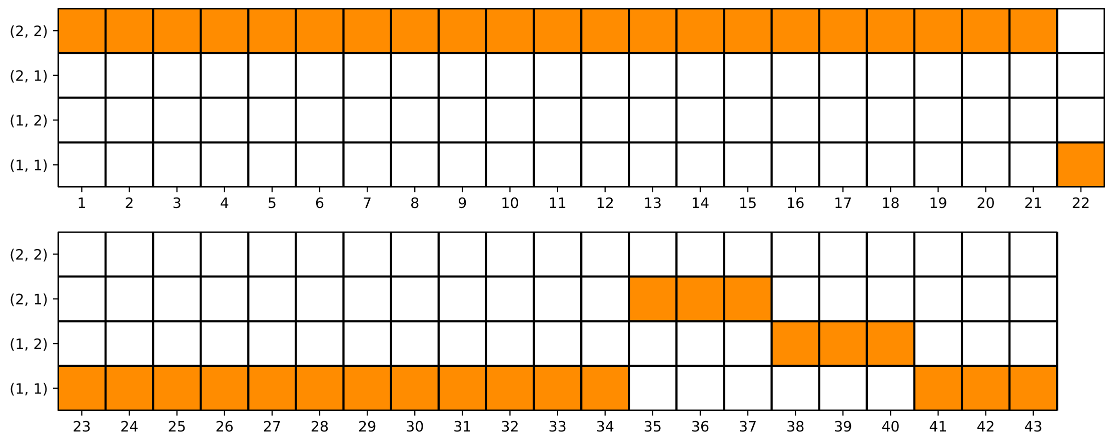


(a) Derived optimal policy for illustrative example I with low discount rate.

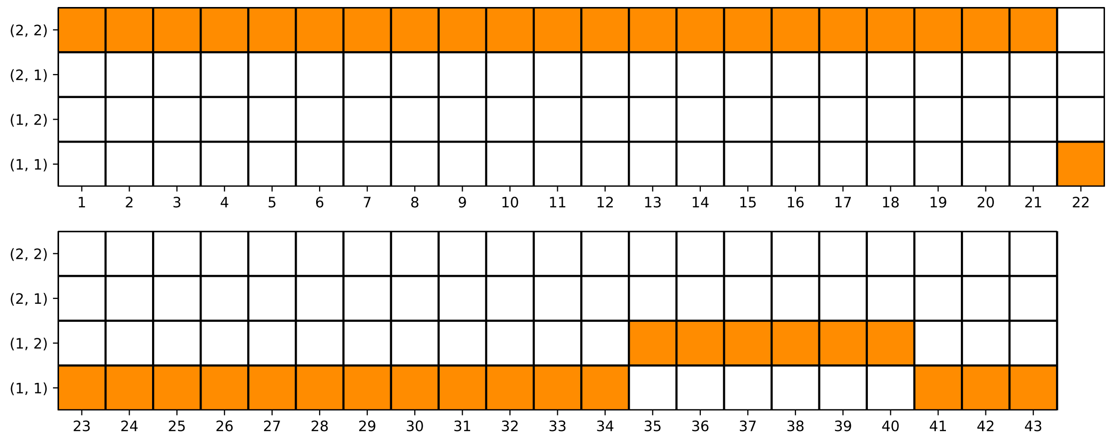


(b) Derived optimal policy for illustrative example I with high discount rate.

Figure 6.13: Derived optimal policies for illustrative example I.

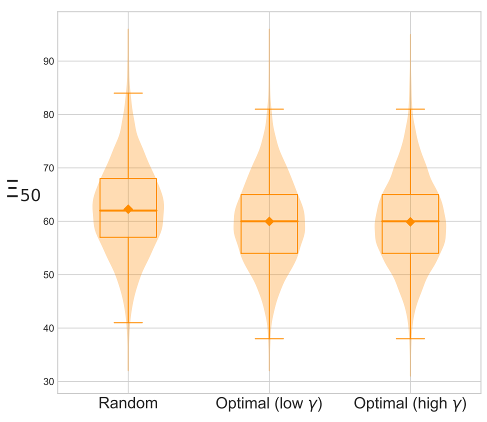


(a) Derived optimal policy for illustrative example II with low discount rate.

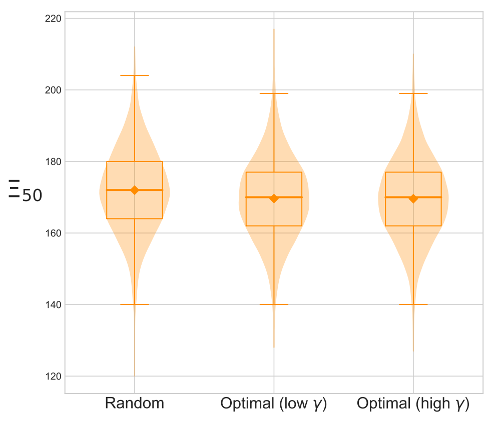


(b) Derived optimal policy for illustrative example II with high discount rate.

Figure 6.14: Derived optimal policies for illustrative example II.



(a) Distributions of Ξ_{50} for the random and optimal policies of illustrative example I.



(b) Distributions of Ξ_{50} for the random and optimal policies of illustrative example II.

dom and optimal policies are equal, and the alternative hypotheses are that the optimal policy yields a lower mean Ξ_{50} value. Very small p-values, 1.536×10^{-83} and 2.322×10^{-89} were obtained for illustrative example I for the low and high discount rates respectively. Similarly for illustrative example II, 9.706×10^{-48} and 1.736×10^{-50} were obtained for low and high discount rates respectively. These indicate that the improvements observed by following an optimal deadlock resolution policy are significant. These improvements are however very small.

Discussion

The results of the two illustrative examples are intriguing. There is evidence to show that using an optimal policy does significantly lower the number of deadlock resolutions required in a simulation run. However these improvements are small. This may indicate, at least for the small illustrative examples given here, that finding optimal deadlock policies may in fact be a fruitless endeavour. However, further research on a wider variety of queueing systems is required in order to conclude whether there are any systems in which the optimal policy yields larger performance improvements.

The method of finding optimal policies discussed in this chapter is not scalable for use on a large number of systems. It also isn't scalable for use on a single system much larger than the illustrative examples here. The reason is because there are a number of steps that need to be completed manually: identifying all deadlocked states, identifying feasible action sets for each state, and running enough simulations to approximate the transition probabilities and rewards with empirical values.

Another method is proposed, though not implemented in this thesis. Reinforcement learning is a subset of unsupervised machine learning, it is a computational way of learning and decision-making through goal seeking means. Comprehensive overviews are given in [158] and [159]. Reinforcement learning, specifically the Q-learning algorithm, can be used to solve an MDP by taking actions and exploring the state space, by interacting with the real or a simulated system. Agents 'visit' states, take actions, receive rewards, and transition to another state according to the transition probabilities. It can be thought of as a simulation of the MDP, however actions are chosen according to some rule parametrised by the value $Q(z, a)$, the expected total reward for choosing action a in state z. These values are updated are more information is gained about the system by interacting with it, according to the following relation:

$$Q(z, a) \longleftarrow (1 - \alpha)Q(z, a) + \alpha \left(\omega(z, z', a) + \gamma \max_{a'} Q(z', a')\right) \tag{6.6}$$

where $0 < \alpha \leq 1$ is the learning rate of the algorithm, dictating how ready the agent is to trust its new information. For a deterministic system a large learning rate is appropriate, and would allow fast convergence of the Q-values. For a stochastic system, such as the deadlock resolution problem, a smaller learning rate would account for the variability, inhibiting the agent from causing the Q-values from fluctuating with the stochasticity of the system.

This reinforcement learning algorithm can be implemented with the Ciw simulation framework, due to its open, modular and extendible nature. One agent would control the actions and update the Q-values whenever the system reaches deadlock, making use of the deadlock detection mechanism in Ciw. In this manner there is no requirement to predefine the state space, and transition probabilities and rewards arise naturally through the interaction with the real or simulated system, eliminating the need to find empirical values through simulation.

For larger systems however, multi-server systems with more than two nodes, finding action sets would still be difficult. As shown in Section 6.3 not all actions lead to completely resolved deadlocks; and at each deadlock a set S of customers must be found such that and S-constrained sequence of unblockages resolves the deadlock. Finding the maximal such S is NP-hard; and there may be a number of choices of S which are not maximal (there may be a number of choices of S which are maximal also). In which case the choice of S becomes part of the action space, further complicating the MDP.

6.5 Summary

This chapter has investigated one method of resolving deadlocks in restricted queueing networks, namely by creating some virtual capacity and allowing a sequence of constrained unblockages such that that virtual capacity is no longer required.

First, sequences of unblockages were defined in terms of a graph theoretical sequence, equivalent to traversing the edges of a directed multigraph and removing edges as they are traversed. It was shown that if unblockages are constrained such that the directed multigraph is Eulerian, then deadlock is resolved sufficiently, where the virtual capacity originally introduced is no longer required. It was also shown that for some systems, single server systems and two node systems, this condition is not required.

By way of an example, it was shown that the deadlock resolution policy, that is the choice of which deadlocked customer is to be placed in virtual capacity, effects which non-deadlocked state the system will cumulate in. As it is reasonable to assume that deadlock resolutions have some associated cost (whether monetary or computational), the natural question then is to ask what is the optimal deadlock resolution policy to minimise costs? Attempts at answering this were given for two illustrative examples by formulating and solving a Markov decision process; though it was shown that the performance of the optimal policies did not differ greatly from that of a random policy. This offers valuable insight from the point of view of deadlock resolution in real or simulated situation: a random approach is adequate.

All code used in the chapter is available at [126] with DOI `10.5281/zenodo.1420393`.

Chapter 7

Modelling Stay Well Plans in Gwent

In this chapter simulation modelling is used to investigate the effects of a new preventative intervention, Stay Well Plans (SWPs), on the demand and workforce requirements of a healthcare system in Gwent. A large focus of this chapter concerns the extrapolation of a data set covering a small geographic area to a larger area, accounting for the differing natures of the areas and populations.

The Chapter is structured as follows:

- Section 7.1 introduces the scenario and the research questions that this chapters sets out to answer.
- Section 7.2 describes the healthcare facilities to be modelled and the possible patient flows between the facilities.
- Section 7.3 describes the population of both the study cohort and of the geographic areas to be modelled, while Section 7.4 describes the 15 classes of patient that will populate the model.
- Section 7.5 uses data analysis to derive the arrival, routing and service parameters of the model.

- Section 7.6 extrapolates the parameters of the previous section to account for other geographic areas with differing characters. This includes adjusting arrival rates to account for different population sizes, adjusting SWP uptake probabilities to account for different deprivation levels, and adjusting routing matrices to account for different proximities to hospitals.
- Section 7.7 validates the model for Newport county.
- Section 7.8 introduces the workforce models that are applied to the demand data produced by the simulation model.
- Section 7.9 gives all demand and workforce results and conclusions, which are summarised in Section 7.10.

A number of data sets were used in this analysis. Table 7.1 summarises the sources of these data sets.

7.1 Introduction

In August 2014, a trial selection of GP practices in the Newport area began offering Stay Well Plans (SWPs) to a subset of their patients who are over 60 years of age. These are plans that are delivered at home, and may include advice on cleanliness and nutrition, advice on where, why and when to access the healthcare system, initial home inspections, and other general help. A detailed sumamry is given in Chapter 1. Another type of Stay Well Plan, Frailty SWPs (FSWPs) are also given to some patients, although they are omitted from this analysis due to the inherently different nature of those patients being selected for FSWPs.

This chapter attempts to answer three questions, using a combination of data analysis and discrete event simulation techniques:

i. **What effects are Stay Well Plans having on how the population of older people in Newport are accessing the healthcare system?**

Description	Source
Comprehensive list of all patients in the study cohort, including date of birth and LSOA.	Provided by the heath board.
List of all patients who were offered or received Stay Well Plans, including plan start date.	Provided by the health board.
Attendance data for ED, general ward admissions, and frailty services, including arrival date, discharge date, and frailty priority.	Provided by the health board.
Local authority community care services data, including care package type, start and end date.	Provided by the health board.
Total hourly demand at the A&E department at Royal Gwent for 2015.	Provided by the health board.
Lower layer super output area boundaries in geojson format.	[117]
Welsh index of multiple deprivation for 2014.	[156]
Mid-2015 population estimates for lower layer super output areas in England and Wales by single year of age and sex.	[118]
Death registration summary statistics for England and Wales 2015.	[116]
Local authority population projections for Wales 2014.	[183]

Table 7.1: Data sources for the analysis.

ii. **What would be the effect on demand and workforce needs of the healthcare system if the plans were to be rolled out to all five counties served by Aneurin Bevan University Health Board?**

iii. **How do these effects, if any, scale with projected population growth?**

Initially it is thought that this analysis will show two things: That patients receiving SWPs access emergency services less often, but are presenting with more severe illnesses and injuries; and that they are accessing non-emergency and preventative healthcare facilities more often, though the amount of time spent there might be shorter. It is believed that, for the first conjecture, the patients that are selected to be offered a SWP are those that are more ill than the rest of the population, thus are presenting with more severe complains. However, due to the effects of the SWPs, they are not presenting for less severe complains as much, and are making use of other non-emergency and preventative services instead. Another aim of the plans is to keep patients healthier at home longer, that is out of residential care.

In this study, data from a study cohort of 12605 people living within Newport county is analysed, over an observation period between 01/04/2014 and 30/08/2017. The model will aim to both evaluate the effect of the SWPs within the study cohort, especially in terms of workforce requirements, and also to extrapolate these findings to investigate the effect of rolling out the plans to all five counties of Gwent.

This extrapolation poses some interesting challenges, as the data from the study cohort does not contain some vital information required to describe the behaviours of the overall population. This is because patients from different counties are believed to have vastly differing behaviours: namely different population sizes across counties, living in extremely different levels of deprivation, and different preferences for hospitals due to differing proximities from homes. Another challenge posed by this work is that the level of granularity of the data, and thus the model and its results, are much lower than that required for accurate workforce planning. In addition, lack of recorded workforce activity ensues inaccurate or incomplete workforce models. Thus a number of reasonable assumptions and adjustments will be made to give sensible measures of workforce needs.

This work will follow the process map illustrated in Figure 7.1. Patient level data from Aneurin Bevan University Health Board (ABUHB) and the local authority, combined with publicly available demographic data, are analysed and model parameters are derived. These parameters are given to a simulation model of the patient flows around the whole system, from which projected demand is found. Some of this projected demand data is then put through a workforce model to find projected health board workforce data. Some is used as input parameters for another simulation model, from which more accurate residential care demand can be found. This is then put through a workforce model to find projected residential care workforce data.

It is important to note that at each step, that is every arrow in Figure 7.1, uncertainty increases. This uncertainty is not always captured numerically in the model results.

A large source of uncertainty is the extrapolation of geographical data from a small study cohort to the whole of Gwent. Figure 7.2 illustrates the steps taken to reasonably extrapolate across geographically different areas. There are three main steps, described in detail in Section 7.6: expanding out to the whole population; considering how deprivation affects behaviour; and considering how hospital proximity affects behaviour.

7.2 The older person's integrated care pathway

A foundation of this work is the network of health care facilities that the older people analysed in this work make use of, and flow within. An initial network was drawn following numerous discussions with healthcare professionals from the ABUHB Workforce Planning department, Public Health Wales, ABUHB Frailty

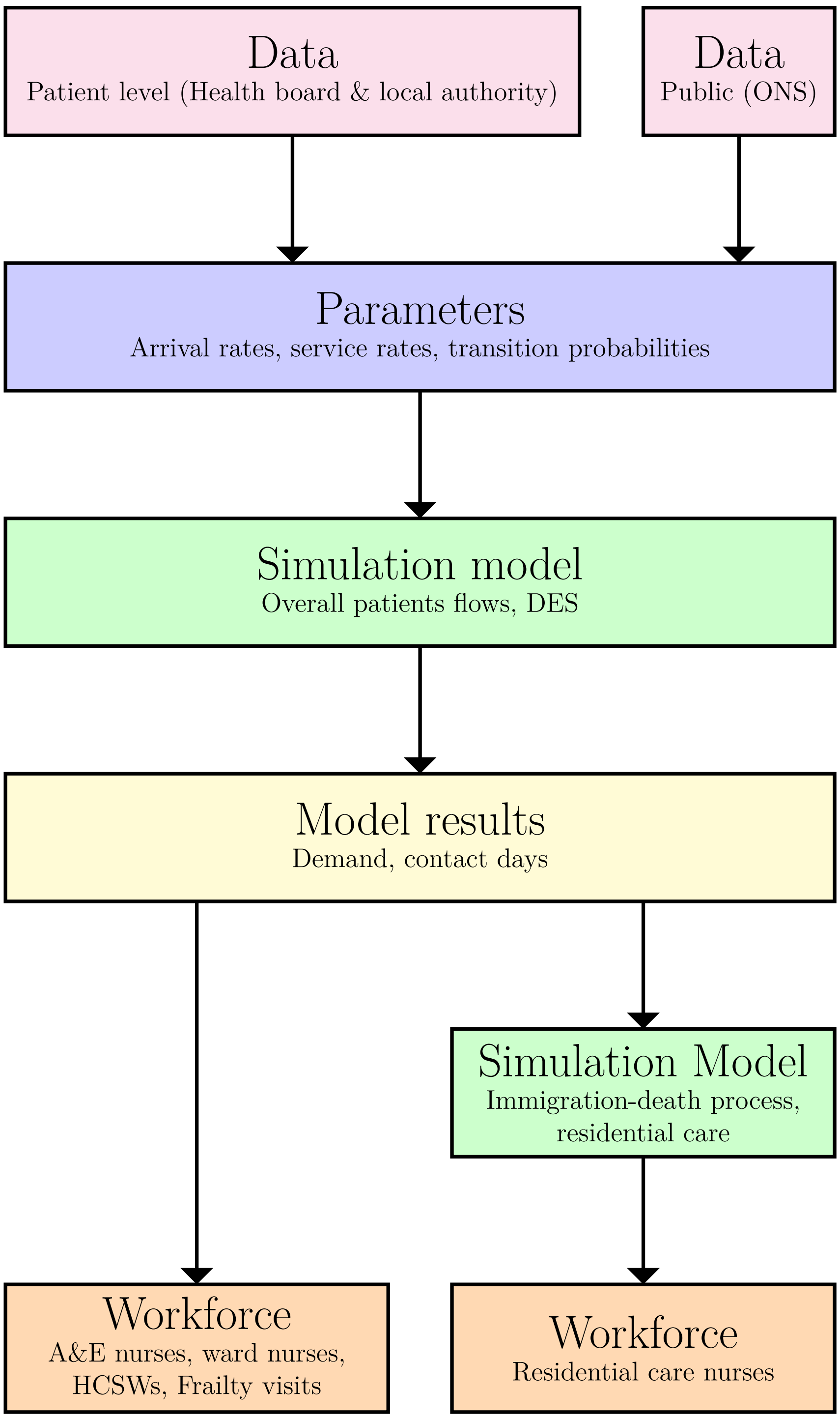

Data
Patient level (Health board & local authority)
Data
Public (ONS)
Parameters
Arrival rates, service rates, transition probabilities
Simulation model
Overall patients flows, DES
Model results
Demand, contact days
Simulation Model
Immigration-death process, residential care
Workforce
A&E nurses, ward nurses, HCSWs, Frailty visits
Workforce
Residential care nurses

Figure 7.1: Process map of the modelling work.

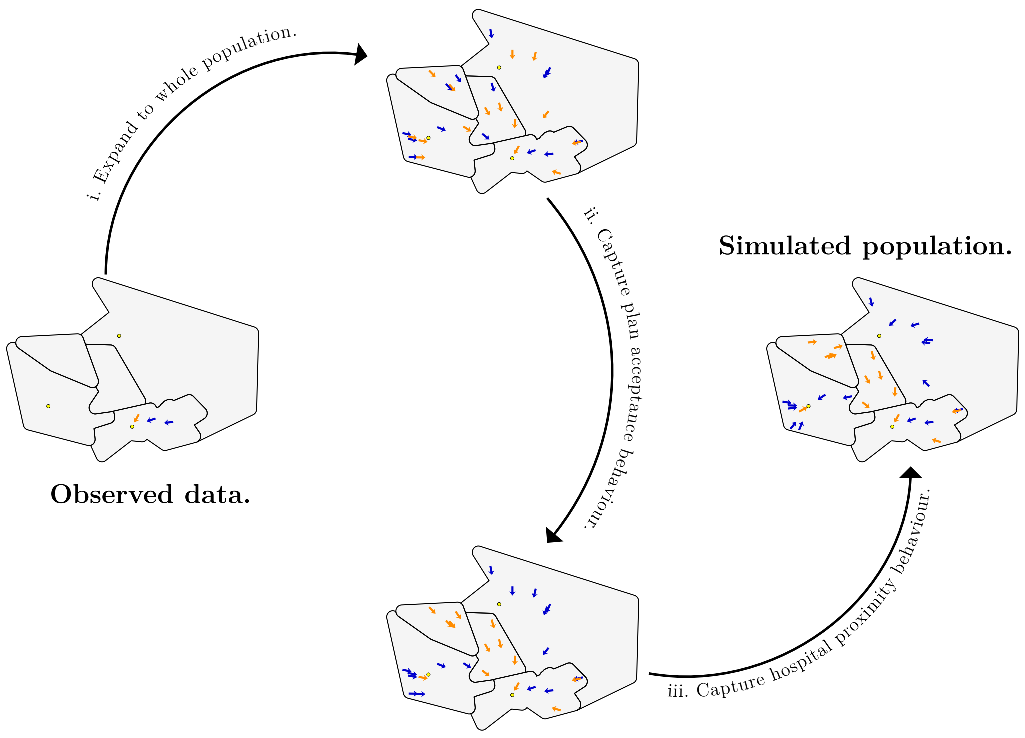

i. Expand to whole population.
ii. Capture plan acceptance behaviour.
iii. Capture hospital proximity behaviour.
Simulated population.
Observed data.

Figure 7.2: Illustration of the geographic data extrapolation carried out in this work.

Department, the ABCi (Aneurin Bevan Continuous Improvement), PeopleToo, and academics at Cardiff University. This hypothesised map of patient flows in shown in Figure 7.3 (The same diagram that was shown and discussed in Section 1.3).

Modelling these patient flows however is limited to the available data. Five data sets were provided, giving information on ED Attendances, General Ward Admissions, Assessed Out records, Frailty episodes, and a SWIFT data set providing local authority community and residential care data. ED attendances across Gwent, which consist of both A&E attendances and MAU (medical assessment unit) attendances at different hospitals, are captured as just one health care facility in this model. Similarly for Assessed Out, which is only captured to record the number of patients leaving the system through that particular route. Frailty episodes are categorised into four separate specialities or priorities:

- **Rapid Medical:** This service is an emergency service, that rapidly responds to medical household emergencies. These are one-time on-the-day services,

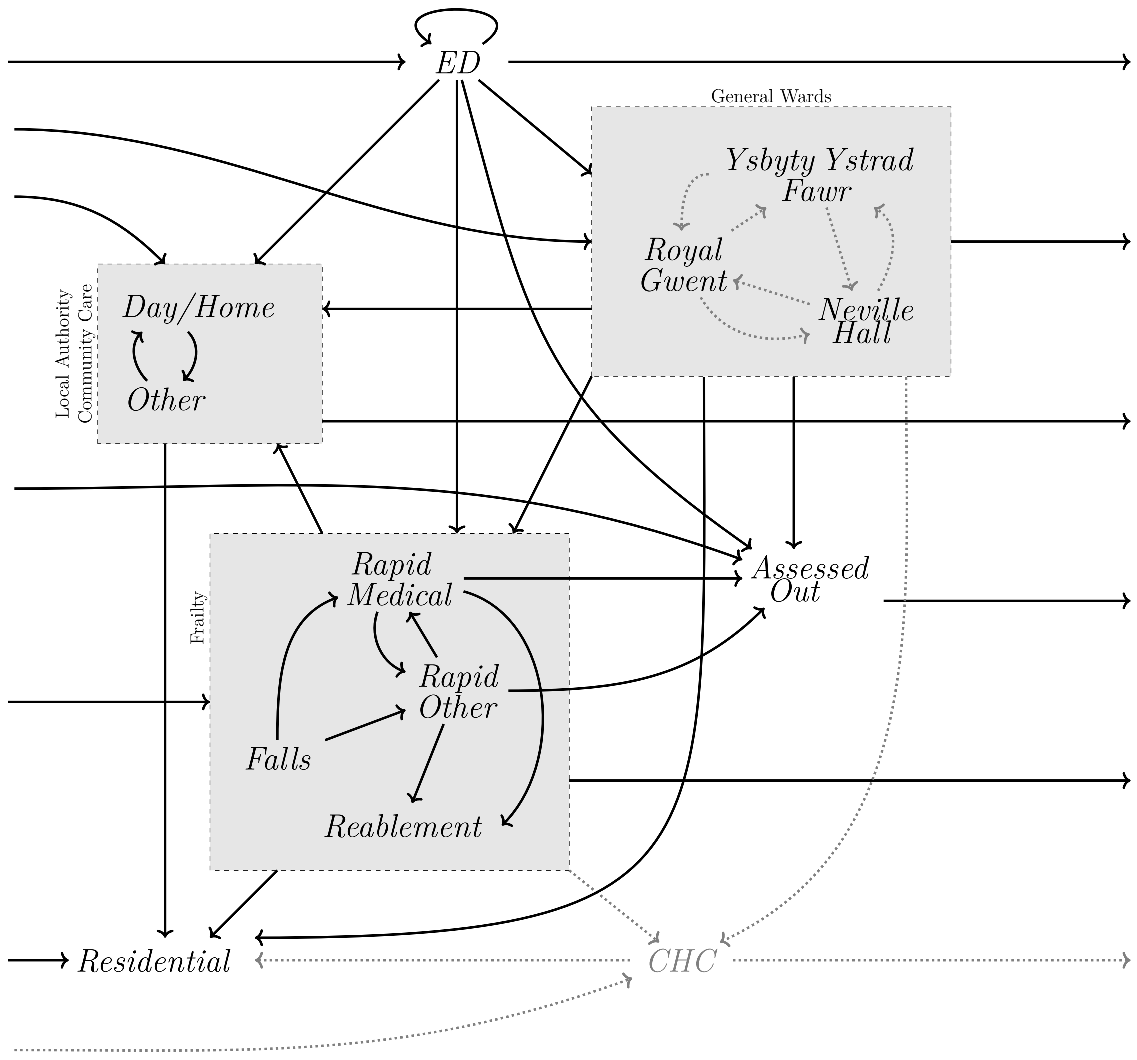


Figure 7.3: Hypothesised map of the older person's integrated care network. Greyed out facilities and routes have been identified as being important to consider, however are not captured in this analysis due to data restrictions. (CHC is Continuing Healthcare services.)

although follow-up appointments may occur.

- **Rapid Other:** This service is an emergency service, that rapidly responds to household emergencies that are not medical in nature. These are one-time on-the-day services, although follow-up appointments may occur.
- **Reablement:** This service provides some rehabilitation after a health care episode. For example this can include physiotherapy, continued visits and check ups, and housing inspections. These episodes tend to be prolonged and continuing care.
- **Falls:** This service provides a combination of the above three, but is specifically for responding to and preventing falls in the home. These services can include preventative measures such as housing inspections, and responsive action to a fall including physiotherapy.

A number of community care packages are provided. Often they are bespoke, and difficult to describe exactly what is involved in the care. From discussions with professionals at PeopleToo and Public Health Wales, these packages were categorised into three types: Home and Day Care, Residential Care, and all other services. Home/Day and other services may be seen as preventative services, as they attempt to keep patients well in the community and at home.

General Ward Admissions may be at three different hospitals, Royal Gwent, Neville Hall, and Ysbyty Ystrad Fawr. Also included in the data set were some admissions to St Woolos, although for the purposes of this study, these have been included as Royal Gwent admissions.

Therefore the following twelve healthcare facilities are those that are analysed:

1. ED Attendance
2. Royal Gwent
3. Neville Hall
4. Ysbyty Ystrad Fawr
5. Assessed Out
6. Frailty - Rapid Medical

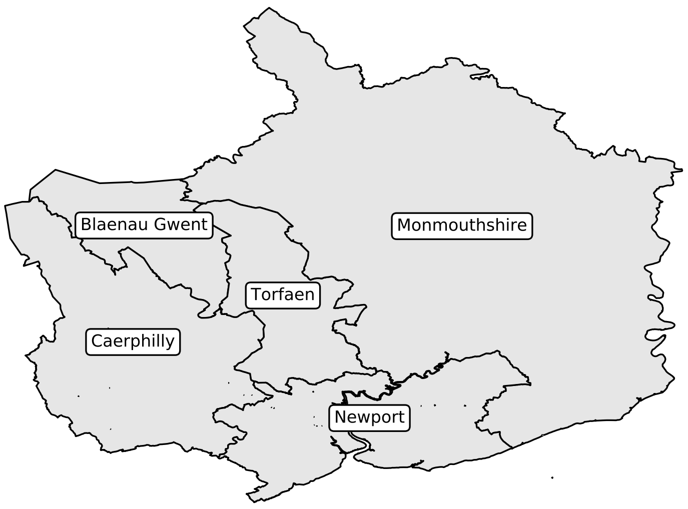


Figure 7.4: Map of the five counties of Gwent.

7. Frailty - Rapid Other
8. Frailty - Reablement
9. Frailty - Falls
10. Community Care - Home/Day Care
11. Community Care - Other
12. Community Care - Residential

The ordering and numbering of the facilities here will be used throughout.

7.3 Population analysis

Gwent is an area of Wales encompassing five counties: Blaenau Gwent, Caerphilly, Monmouthshire, Newport and Torfaen. A map of Gwent labelling these counties is given in Figure 7.4 for reference. From ONS mid 2015 estimates, the populations of people over 60 years of age for each county is displayed in Table 7.2.

A study cohort, a subset of the Newport population of size 12605, is studied. This corresponds to 37.97% of the estimated population of people over 60 years olds living in Newport county, and 8.61% of the estimated population of people over 60 years old living in the whole of Gwent.

County	Over 60
Blaenau Gwent	17557
Caerphilly	43864
Monmouthshire	28293
Newport	33248
Torfaen	23624

Table 7.2: Mid 2015 estimates of the populations over 60 for each county of Gwent.

	All Patients	Non-SWP Patients	SWP Patients
Bellevue	1934	1856	78
Bryngwyn	223	72	151
Eveswell	914	914	0
Gaer Medical Centre	1	0	1
Grange Clinic	1643	1602	41
Malpas Brook	1508	1461	47
Richmond	1240	1240	0
Ringland	1213	1157	56
Rogerstone	2274	2171	103
Rugby	1145	1105	40
St Davids	401	103	298

Table 7.3: Number of patients in the study cohort registered at each GP practice, and whether or not they received a SWP.

SWPs were trialled on this cohort over a three year period (01/04/2014 to 30/08/2017). Table 7.3 gives the counts of people of the study cohort who are registered at each GP practice. The table also details whether they received a SWP or not during the three year period. The table highlights a fact reported by the heath board, that GP practices took different strategies when choosing patients to receive a SWP. It is known that some GP practices chose patients to be offered SWPs by age, some by using a bespoke risk stratification tool, and some on a one to one basis. For example a majority of patients from St Davids and Bryngwyn received plans. Both these practices offered plans to all patients over 60 years of age. However only a minority of patients received plans from other GP practices. Some practices, Eveswell and Richmond, did not implement the plans during the study period. Only a single patient from Gaer Medical Centre is included in the study cohort, this is probably due to the patient changing address and GP practice during the observation period.

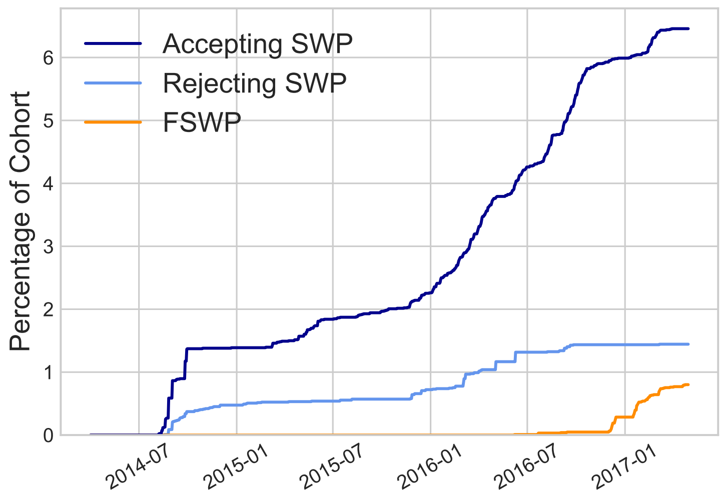


Figure 7.5: Uptake of SWPs over the observation period.

Figure 7.5, showing the uptake of SWPs over time, indicates that the uptake of these plans over time was not consistent either. Four main periods of differing behaviour can be seen. First a short period is seen where the number of patients receiving SWPs increases dramatically, this is around July 2014 when plans were first offered. Then there is a period in which uptake was stagnant, until January 2016. For the next year the number receiving increases steadily, until December 2016 where FSWPs are initially introduced. Interestingly, with the introduction of FSWPs, the number of people rejecting plans ceases to increase.

Looking at gender, the study cohort consists of 54.88% females, while of those receiving a SWP 59.93% are female. This does not evidence that gender affects SWPs' take up rate. Similarly the age distributions of those receiving and not receiving SWPs is investigated, shown in Figure 7.6, and shows no differences between the two groups.

In 2014 the ONS produced 26 year population projections for each county in the UK. The projections for the counties of Gwent, for the population of people over 60 years old, from 2015, are summarised in Figure 7.7. The two upper subplots show the absolute projections. Caerphilly, the county with the largest population of over 60s, is projected to increase the most in terms of absolute numbers. Counting all

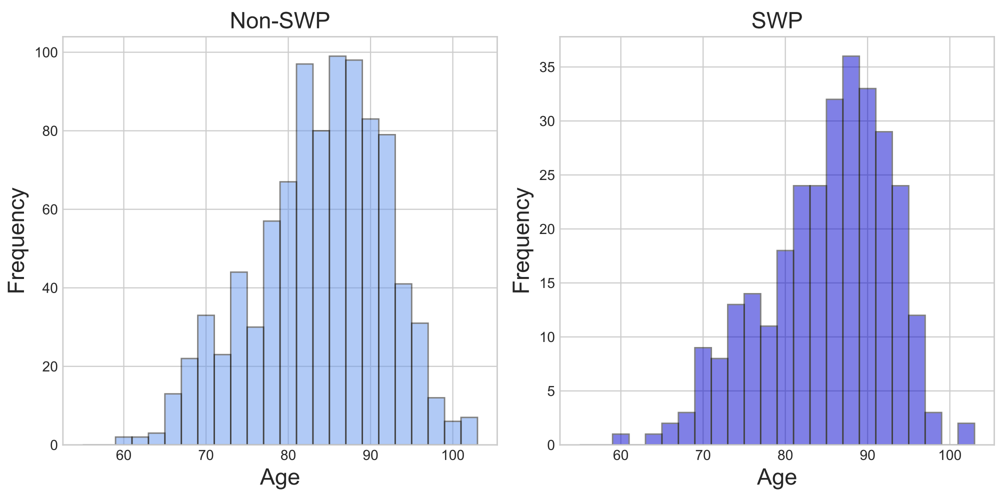


Figure 7.6: Age distributions of those patients receiving and not receiving SWPs.

five counties together as Gwent, the number of over 60s are expected to increase to nearly 200,000 by 2039. The shape of the projections curves are interesting, the growth in population is expected to ease by around 2030, with a fairly stable population size for the next ten years. The two lower subplots show the percentage differences. Monmouthshire is expected to have the largest percentage increase of over 60s, with an increase of over 40%. This is understandable, with Monmouthshire seen an affluent rural county where people might move to retire. Blaenau Gwent, the most deprived of the counties, has the smallest projected percentage increase. The lower right subplot shows the yearly expected percentage change, or the rate of change of the populations. All counties show the same trend, with peak growth at around the year 2023, and a slower rate of growth thereafter.

7.4 Patient flows model

In order to evaluate the effect of the SWPs on the demand and use of the healthcare system in Gwent, the system is modelled as a queueing network. As we are only concerned with the demand at each facility, infinite capacities are assumed for each facility. This eliminates the possibilities of any model deadlocks that are known not to occur in reality.

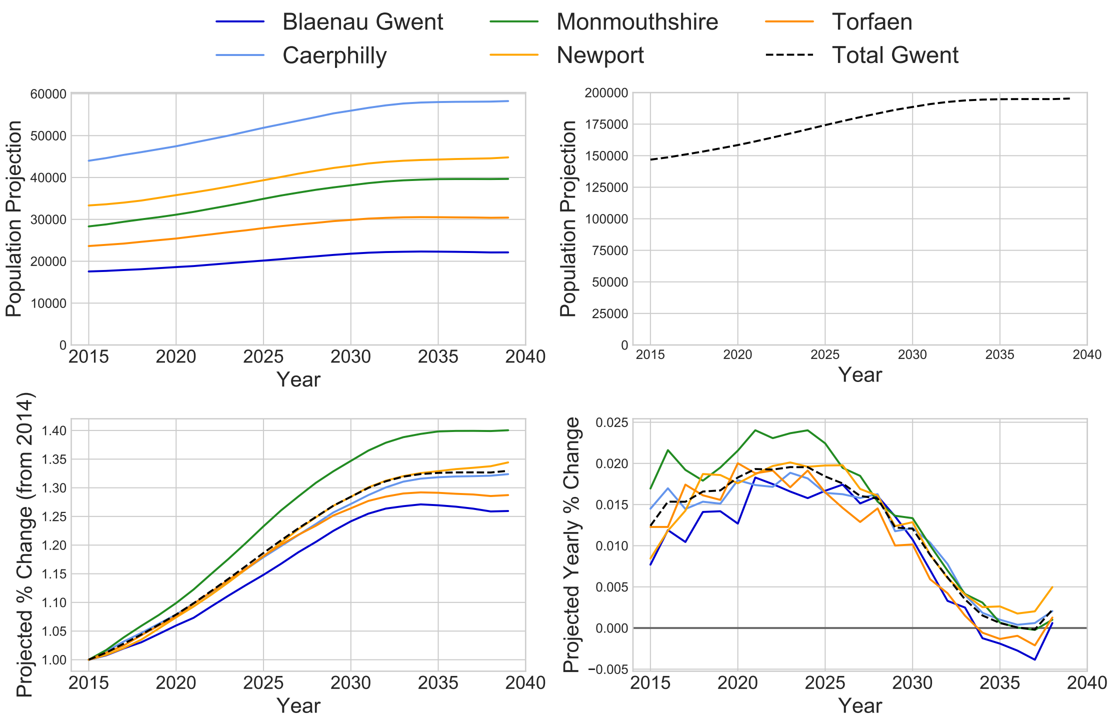


Figure 7.7: Summary of population projections for the counties of Gwent.

The population is modelled as three classes of patient, who may use the healthcare system in very different ways:

- **Class A**: Patients who are receiving a SWP. These correspond to those patients that have accepted a SWP.
- **Class R**: Patients who are eligible for a SWP but have not received a plan. These correspond to those patients who rejected a SWP.
- **Class N**: Patients who are well enough not to be eligible for a SWP.

Patients receiving Frailty SWPs are omitted from the model. Given the findings in Section 7.3 that GP practices used different methods of selecting patients to be offered SWPs, the lack of confidence in the risk stratification tool, and that gender and age do not effect this either, the model will assume that SWPs are offered to all patients eligible (class R and class A combined). Thus the model assumes that all eligible patients correspond to all patients in the data set who were offered a plan. The proportions p_o, the probability of being offered a SWP, p_a, the probability of accepting a SWP once offered, and p_f the probability of receiving an FSWP, will define the proportions of A, R and N.

As of the end of the observation period, 30/4/2017, out of a cohort size of 12605, 128 patients had received a Frailty SWP, 917 patients had received a SWP, and 184 had rejected a SWP. This corresponds to a probability of receiving a Frailty SWP of $p_f = 0.0102$, a probability of being offered a SWP of $p_o = 0.0882$, and a probability of accepting a SWP of $p_a = 0.8329$. These three parameters will be used throughout to define the population.

7.5 Model parameters

In order to fully define the simulation model that will be used to evaluate the effect of SWPs, the following information is required: the duration of time spent at each facility, the external arrival rates to each facility, and analysis on the pathways between facilities. In this section, these parameters will be found using only the final year of the observation period, as this period had the most SWPs (see Figure 7.5), in order to more evaluate the effect of those plans.

Figure 7.3 displays all possible routes that patients may take once in the system. Only those routings in solid black lines are considered, routes in grey dotted lines are omitted from the model as the data could not capture these routes. The data set gave detailed routing information for patients exiting ED, and thus could be read directly. For all other facilities, this analysis assumes that patients were transferred between two facilities if the date of admittance to the second facility was within two days of the date of discharge for the first facility. Table 7.4 summarises this analysis. The percentage of patients leaving each facility that are routed to each relevant destination are given for each class of patient, A, R and N. A χ^2 test was carried out to determine whether the differences in routing between these three classes of patient was significant. The null hypothesis is that there is no significant difference in the routing probabilities between the three classes, and any observed

Facility	Destination	A %	R %	N %	p-Value
ED Attendance	ED Attendance	0.32	0.00	0.16	0.780 06
ED Attendance	Assessed Out	0.00	0.00	0.29	0.588 42
ED Attendance	Neville Hall	0.64	0.00	0.49	0.805 99
ED Attendance	Royal Gwent	38.59	44.07	31.74	0.009 22
ED Attendance	Ysbyty Ystrad Fawr	0.00	0.00	0.04	0.927 19
ED Attendance	Leave	60.45	55.93	67.28	0.013 23
Community Care - Home/Day	Community Care - Other	2.76	3.85	6.25	0.409 04
Community Care - Home/Day	Community Care - Residential	0.69	3.85	2.08	0.399 23
Community Care - Home/Day	Leave	96.55	92.31	91.67	0.242 53
Community Care - Other	Community Care - Home/Day	14.58	20.00	16.67	0.934 06
Community Care - Other	Community Care - Residential	0.00	0.00	3.33	0.408 97
Community Care - Other	Leave	85.42	80.00	80.00	0.809 16
Royal Gwent	Community Care - Home/Day	8.14	16.92	0.85	5.9×10^{-27}
Royal Gwent	Community Care - Other	0.00	0.00	0.06	0.903 31
Royal Gwent	Frailty - Rapid Medical	1.02	0.00	0.90	0.726 14
Royal Gwent	Frailty - Rapid Other	0.34	0.00	1.07	0.349 94
Royal Gwent	Frailty - Reablement	1.36	0.00	2.71	0.163 21
Royal Gwent	Leave	89.15	83.08	94.41	1.2×10^{-5}
Frailty - Falls	Community Care - Other	14.29	0.00	1.04	0.045 86
Frailty - Falls	Frailty - Rapid Medical	0.00	0.00	1.04	0.953 78
Frailty - Falls	Frailty - Rapid Other	0.00	0.00	1.04	0.953 78
Frailty - Falls	Leave	85.71	100.00	96.88	0.316 87
Frailty - Rapid Medical	Assessed Out	3.33	0.00	2.12	0.895 78
Frailty - Rapid Medical	Frailty - Rapid Other	0.00	0.00	0.53	0.918 47
Frailty - Rapid Medical	Frailty - Reablement	0.00	0.00	2.65	0.648 51
Frailty - Rapid Medical	Leave	96.67	100.00	94.71	0.854 06
Frailty - Reablement	Community Care - Home/Day	9.68	20.00	2.11	0.003 78
Frailty - Reablement	Leave	90.32	80.00	97.89	0.003 78

Table 7.4: Observed transition rates between facilities, with χ^2 test results.

differences is because of sampling. The p-values of this test are reported in the table.

From this table we see that four routes are significantly effected by SWPs:

- ED to Royal Gwent: Class R are more likely to be admitted to Royal Gwent from ED than Classes A, who are more likely to be admitted than Class N.
- Royal Gwent to Community Care - Home/Day: Class R are more likely to be referred to Home and Day Community Care services after a spell at Royal Gwent than Class A, who are more likely to be referred than Class N.
- Royal Gwent to Leave: Class R are less likely to be sent home needed no more services after a spell at Royal Gwent than Class A, who are less likely to be sent home than Class N.
- Frailty - Reablement to Community Care - Home/Day: Class R are more likely

to be referred to Home and Day Community Care services after Reablement Frailty services than Class A, who are more likely to be referred than Class N.

These all show that patients who are well enough not to be offered a SWP (Class N) are the least likely to be referred elsewhere after being treated at a facility, confirming that these patients are healthier than the other classes; and that those receiving SWPs (Class A) are less likely to be referred elsewhere than those rejecting SWPs (Class R), suggesting that SWPs are keeping patients healthier than otherwise.

The routing percentages found in Table 7.4 can be summarised in transition matrices, where the rows represent the facility and the columns represent the destination, with row and column numbers representing the numbered facilities from Section 7.2. R_N (Equation 7.1), R_R (Equation 7.2) and R_A (Equation 7.3) correspond to the transition matrices for patients of Class N, R and A respectively.

$$R_N = \begin{pmatrix} 0.0018 & 0.3275 & 0.005 & 0.0004 & 0.0025 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0089 & 0.0094 & 0.0244 & 0.0 & 0.0235 & 0.0005 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0226 & 0.0 & 0.0045 & 0.0226 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0241 & 0.012 & 0.0 & 0.0361 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0288 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0095 & 0.0095 & 0.0 & 0.0 & 0.0 & 0.019 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0412 & 0.015 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1566 & 0.0 & 0.012 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \tag{7.1}$$

$$R_R = \begin{pmatrix}
0.0018 & 0.3275 & 0.005 & 0.0004 & 0.0025 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0089 & 0.0094 & 0.0244 & 0.0 & 0.0235 & 0.0005 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0226 & 0.0 & 0.0045 & 0.0226 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0241 & 0.012 & 0.0 & 0.0361 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0288 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0095 & 0.0095 & 0.0 & 0.0 & 0.0 & 0.019 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0412 & 0.015 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1566 & 0.0 & 0.012 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{pmatrix} \tag{7.2}$$

$$R_A = \begin{pmatrix}
0.0032 & 0.3859 & 0.0064 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0102 & 0.0034 & 0.0136 & 0.0 & 0.0814 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0333 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0968 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1429 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0276 & 0.0069 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1458 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{pmatrix} \tag{7.3}$$

External arrival rates and distributions are determined by filtering out of the data set any records corresponding to referrals from other facilities. Figure 7.8 shows the distribution of the inter-arrival times of the external arrivals to each facility, by patient class.

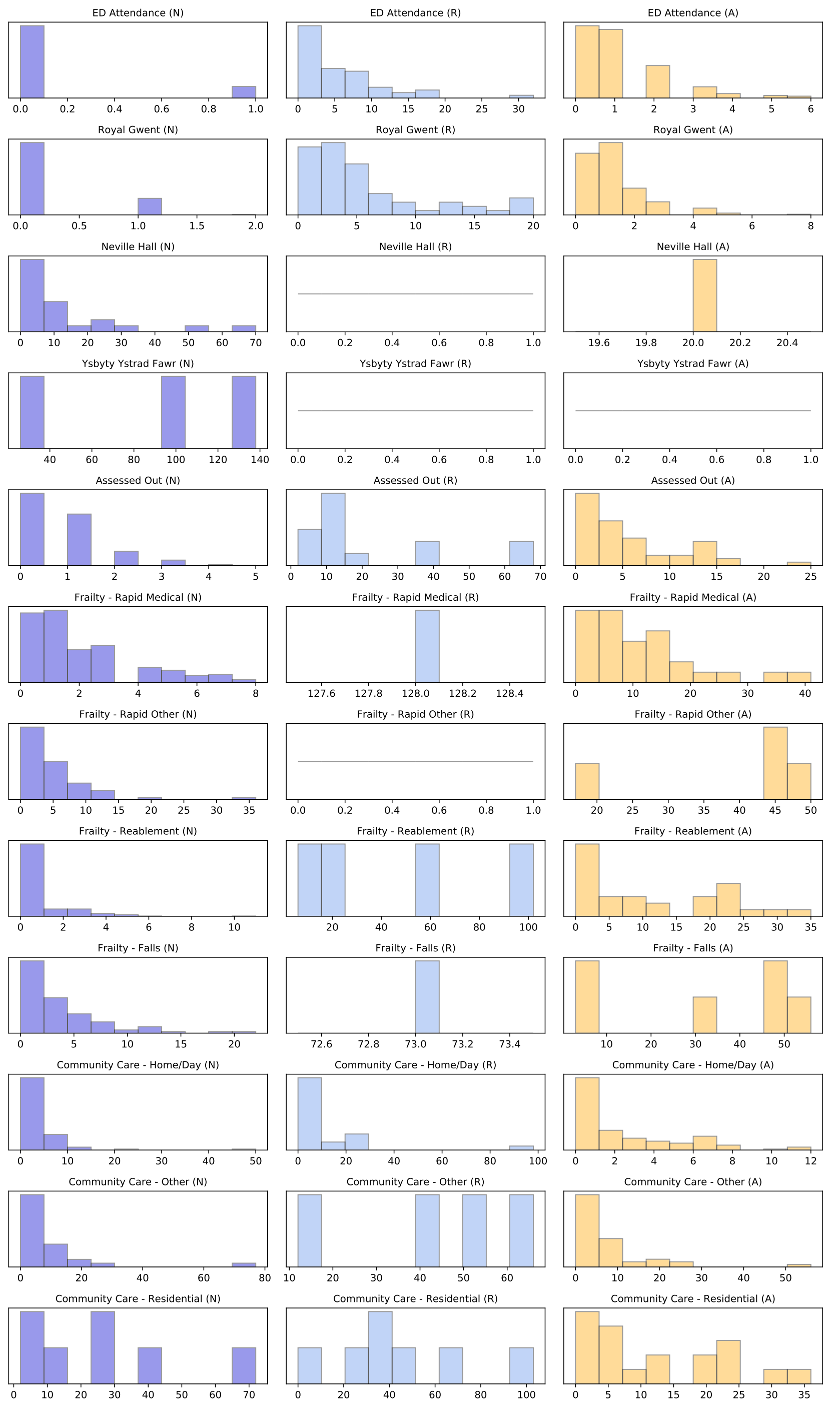


Figure 7.8: Inter-arrival time histograms for each patient class in days.

Many facilities show no or very small numbers of observations. Those facilities that have larger numbers of observations follow rough exponential distributions. Exponential distributions are common and appropriate distributions to use to model arrival distributions, therefore the model will assume exponential distributions for the external arrivals of every facility and customer class.

The modelled arrival rates for the study cohort, for each facility f, are given by

$$\lambda_{(N,f)} = \tilde{\lambda}_f \tag{7.4}$$

$$\lambda_{(R,f)} = \tilde{\lambda}_f \psi_f \left(\frac{p_o \left(1 - p_a\right)}{1 - p_o} \right) \tag{7.5}$$

$$\lambda_{(A,f)} = \tilde{\lambda}_f \psi_f \gamma_f \left(\frac{p_o p_a}{1 - p_o} \right) \tag{7.6}$$

where $\tilde{\lambda}_f$ is the observed external arrival rate for Class N patients to facility f; ψ_f is the effect on the arrival rate to facility f of not being well enough to be offered a SWP; and γ_f is the effect on the arrival rate to facility f of receiving a SWP. These transforms are used in place of the observed values in order to preserve the effects on the arrival rates of belonging to the different patient classes when the rates are scaled up to the different population sizes of different counties. This will become clearer in Section 7.6.

The values for ψ_f and γ_f are calculated by rearranging Equations 7.5 and 7.6, and substituting in the observed values of $\lambda_{(R,f)}$ and $\lambda_{(A,f)}$. That is

$$\psi_f = \frac{\tilde{\lambda}_{(R,f)}}{\tilde{\lambda}_f} \left(\frac{1 - p_o}{p_o \left(1 - p_a\right)} \right) \tag{7.7}$$

$$\gamma_f = \frac{\tilde{\lambda}_{(A,f)}}{\tilde{\lambda}_f \psi_f} \left(\frac{1 - p_o}{p_o p_a} \right) \tag{7.8}$$

Facility (f)	$\tilde{\lambda}_f$	γ_f	ψ_f
ED Attendance	6.684932	1.054285	1.494975
Royal Gwent	2.690411	0.902944	2.392456
Neville Hall	0.032877	1.000000	1.000000
Ysbyty Ystrad Fawr	0.008219	1.000000	1.000000
Assessed Out	1.243836	0.816950	1.906531
Frailty - Rapid Medical	0.465753	2.708833	0.727366
Frailty - Rapid Other	0.158904	0.855563	1.000000
Frailty - Reablement	0.879452	1.083533	0.963023
Frailty - Falls	0.263014	0.702290	1.288043
Community Care - Home/Day	0.186301	1.666974	11.819693
Community Care - Other	0.054795	2.106870	12.365217
Community Care - Residential	0.010959	0.635405	92.739130

Table 7.5: Calculated values for $\tilde{\lambda}_f$, ψ_f, and γ_f for each facility f.

Table 7.5 gives the calculated values for $\tilde{\lambda}_f$, ψ_f and γ_f, for each facility f. Where there is not enough data to find the observed arrival rate, it is assumed that $\psi_f = \gamma_f = 1$.

Finally we look at the duration of time spent at each facility. For general ward admissions that corresponds to lengths of stay, and frailty and community care services, this corresponds to the duration of the care package received. Service time distributions for each facility are given in Figure 7.9. Similarly to the inter-arrival time distributions, it is appropriate to model these as exponential distributions. As the model assumes infinite capacities for each facility this assumption won't cause losses in accuracy.

Table 7.6 gives the calculated values for the service rates $\mu_{(N,f)}$, $\mu_{(R,f)}$ and $\mu_{(A,f)}$, for each facility f for patient classes N, R and A respectively. For the three general ward hospitals considered, the average service rate was taken across the hospitals. It is reasonable to assume that all hospitals have similar lengths of stay, and this overcomes any problems with lack of data for Ysbyty Ystrad Fawr and Neville Hall (the lack of data is due to the nature of the study cohort only residing in Newport county, this is discussed further in Section 7.6). The model will assume that patients spend one full day at ED, no time at Assessed Out, and and infinite amount of time

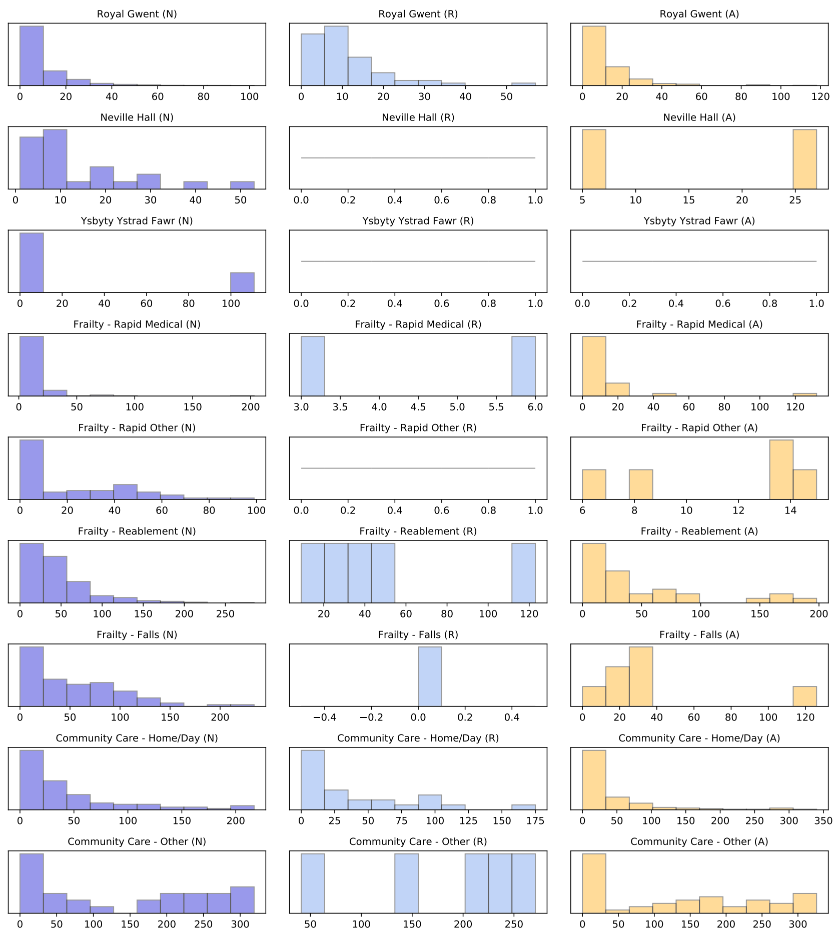


Figure 7.9: Service time histograms for each patient class in days.

Facility (f)	$\mu_{(N,f)}$	$\mu_{(R,f)}$	$\mu_{(A,f)}$
Royal Gwent	0.098527	0.0940666	0.085418
Neville Hall	0.098527	0.0940666	0.085418
Ysbyty Ystrad Fawr	0.098527	0.0940666	0.085418
Frailty - Rapid Medical	0.083113	0.222222	0.078534
Frailty - Rapid Other	0.043357	0.0447198	0.087719
Frailty - Reablement	0.022569	0.0215517	0.020833
Frailty - Falls	0.019150	0.0198977	0.026515
Community Care - Home/Day	0.022357	0.0259222	0.023406
Community Care - Other	0.007339	0.00559284	0.007651

Table 7.6: Calculated values for $\mu_{(N,f)}$, $\mu_{(R,f)}$ and $\mu_{(A,f)}$, for each facility f.

in residential care (that is they are in a residential care home for life).

7.6 Geographic adjustments

The analysis so far has concentrated on the study cohort itself. This is a cohort of 12605 individuals living within Newport county. The purpose of this model is to capture the demand across the whole of Gwent. If it were reasonable to assume that the study cohort behaves similarly to the rest of Newport, then the arrival rates to facilities can scale up, and all other parameters can remain the same. However homogeneous behaviour across all five counties in Gwent cannot be assumed.

The model aims to capture three main differences in behaviour across the five counties, summarised in Figure 7.2. These are:

(i) The arrival rates of individuals from each county to each facility. These are inherently affected by the sizes of the population of over 60 year olds.

(ii) The probability of accepting a plan, p_a. It is thought that deprivation levels of an individual's place of residence can affect the likelihood of accepting a SWP once offered. From Equations 7.4, 7.5 and 7.6 we know that p_a also has an influence on the arrival rates, thus these will be different from county to county.

(iii) The arrival rates and transition probabilities to the three hospitals. In many cases patients or healthcare professionals have a choice of which hospital be admitted or to admit a patient to. This will be affected by the proximity of the hospital to the individual's place of residence.

When writing down the model, the index c will be used to indicate that the parameter varies for each of the five counties; Blaenau Gwent, Caerphilly, Monmouthshire, Newport, and Torfaen. The index s in place of c will indicate the parameter associated with the study cohort. Furthermore, let $|C|$ denote the size of each of the counties, and $|S|$ denote the size of the study cohort, $|S| = 12605$.

(i) Arrival rate scaling

The arrival rates calculated in Equations 7.4, 7.5 and 7.6 are valid for the study cohort, however scaled up arrival rates are required for each each full county c. A simple scaling factor is used to multiply with each arrival rate $\lambda_{(N,f,c)}$, $\lambda_{(R,f,c)}$ and $\lambda_{(A,f,c)}$. for each county c they will be multiplied by $\frac{|C|}{|S|}$. That is county c has a population $\frac{|C|}{|S|}$ times larger than the study cohort, and thus the arrival rate of patients from that county will be $\frac{|C|}{|S|}$ times larger than the arrival rate for the study cohort.

(ii) Probability of accepting a SWP

It is thought that deprivation levels of an individual's place of residence can affect the likelihood of accepting a SWP once offered. The accepted measure of deprivation used in Wales is WIMD (the Welsh Index of Multiple Deprivation), which rank small geographical areas of Wales called LSOAs (Lower Super Output Areas) by deprivation levels. WIMD rankings go from 1 (the most deprived) to 1909 (the least deprived). Gwent is renowned for covering an extreme range of deprivation levels. In fact the LSOAs with the lowest and highest WIMD scores in Wales are

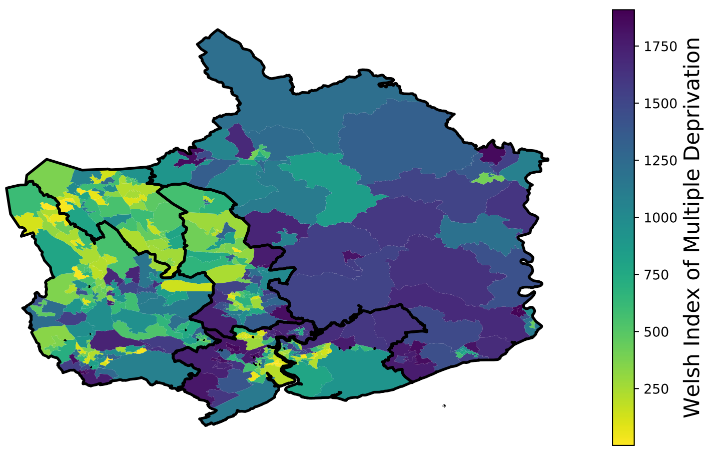


Figure 7.10: Map of deprivation scores by the LSOAs served by ABUHB.

both located in Gwent and served by ABUHB, St. James 3 in Caerphilly is the most deprived, while St. Kingsmark 1 in Monmouthshire is the least deprived.

Figure 7.10 maps the WIMD score for each LSOA in Gwent. It is clear from this plot that each county has a very different character, with Monmouthshire being much less deprived than the other counties, and Blaenau Gwent exhibiting the most deprivation.

Equally important to the analysis is the population of patients over 60 years of age residing in each county. Figure 7.11 maps the population of people over 60 years of age for each LSOA in Gwent. Again this reinforces that each county is distinct, with more elderly people residing in Monmouthshire compared to all other counties.

In order to asses the impact of deprivation on the uptake of SWPs, analysis is undertaken on the proportion of the study cohort offered and accepting SWPs, comparing against the WIMD scores of the patients' place of residence. Figure 7.12 shows the relationship between WIMD score and the observed p_a of the study cohort.

A logistic regression is carried out, in order to predict the probability of accepting a plan p_a from the WIMD score of a patient w. This is also shown in Figure 7.12 as a predicted line. There seems to be a positive correlation between WIMD score and

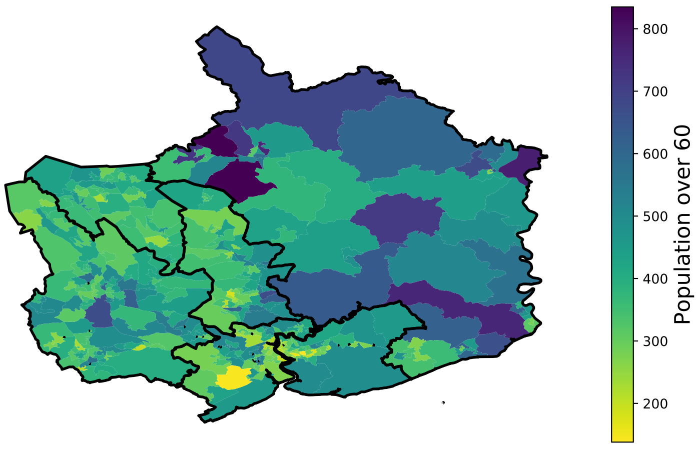


Figure 7.11: Map of the population of individuals over 60 years of age by the LSOAs served by ABUHB.

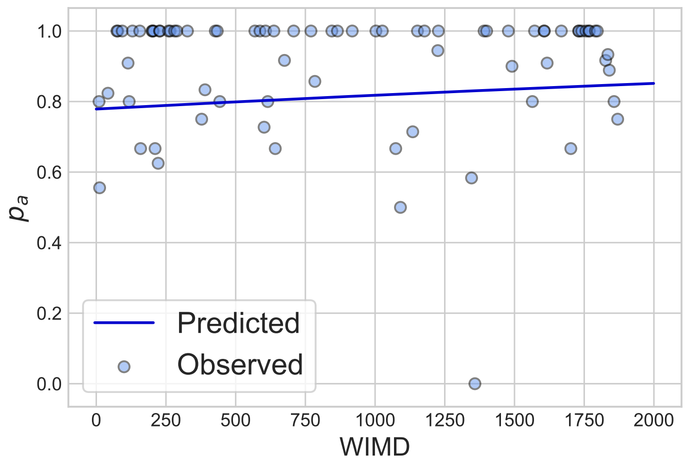


Figure 7.12: Scatter plot of WIMD score against observed probability of accepting a SWP, along with logistic regression fit.

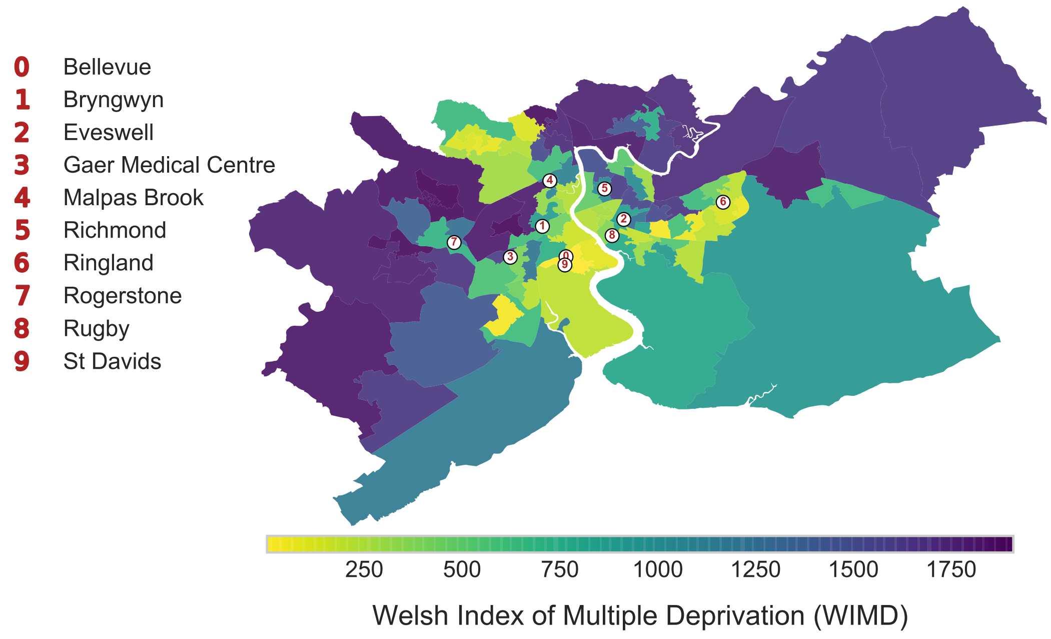


Figure 7.13: Locations of the concerned GP practices in Newport county, overlayed onto a map coloured by WIMD score.

acceptance probability, that is the more deprived an area, the less likely patients from that area are to accept a SWP. The resulting model is given in Equation 7.9.

$$p_a(x) = \frac{1}{1 + e^{1.25836 - 0.00024x}} \tag{7.9}$$

Note for all patients registered to St Davids GP practice and the Bryngwyn GP practice there were missing values for their postcode, and thus deprivation levels of their places of residence could not be found. They were replaced with the median deprivation values in the model. Figure 7.13 shows the locations of the GP practices on a map coloured by WIMD score. It can be seen that both St Davids (9) and Bryngwyn (1) are located in particularly diverse areas in terms of deprivation. Thus it can be assumed that patients registered here will have diverse deprivation scores associated with their home towns, justifying the interpolation and use of the median for these data points in the logistic regression model.

Using this logistic regression model distinct p_a values can be predicted for each LSOA. These are then aggregated by population of over 60s, over the five counties,

	Predicted p_a
Blaenau Gwent	0.799055
Caerphilly	0.809835
Monmouthshire	0.829081
Newport	0.816273
Torfaen	0.813870

Table 7.7: Predicted values of $p_{(a,c)}$ for each county c, using the aggregated results of the logistic regression model described in Equation 7.9.

to obtain specific $p_{(a,c)}$ for each county c. These values are summarised in Table 7.7. Thus, combining the scaling factor discussed in the previous section, and these predicted $p_{(a,c)}$'s, updated arrival rates are proposed in Equations 7.10, 7.11 and 7.12:

$$\lambda_{(N,f,c)} = \tilde{\lambda}_f \left(\frac{|C|}{|S|}\right) \tag{7.10}$$

$$\lambda_{(R,f,c)} = \tilde{\lambda}_f \psi_f \left(\frac{p_o \left(1 - p_{(a,c)}\right)}{1 - p_o}\right) \left(\frac{|C|}{|S|}\right) \tag{7.11}$$

$$\lambda_{(A,f,c)} = \tilde{\lambda}_f \psi_f \gamma_f \left(\frac{p_o p_{(a,c)}}{1 - p_o}\right) \left(\frac{|C|}{|S|}\right) \tag{7.12}$$

(iii) Proximity analysis of hospitals

The largest discrepancy between the parameters of the study cohort and the required parameters of the model for each county c is the arrivals and routing to the three hospitals Royal Gwent, Neville Hall, and Ysbyty Ystrad Fawr. Royal Gwent, located in Newport, is the most popular hospital for patients in the study cohort to attend, with 98.5% of all general ward admissions going here. Neville Hall, located in Abergavenny in North Monmouthshire, far away from where the study cohort live, consists of 1.2% of all general ward admissions by the study cohort. Finally Ysbyty Ystrad Fawr is a smaller hospital in Ystrad Mynach in Caerphilly, and only 0.3% of general ward admissions were at this hospital.

Proximity of place of residence to the hospital is an important factor in which hospital a patient will be admitted to. As motivation to demonstrate that this is important to consider, consider that primary care in Gwent is split into 12 GP clusters:

- Caerphilly South
- Caerphilly East
- Caerphilly North
- Blaenau Gwent East
- Blaenau Gwent West
- Monmouthshire North
- Monmouthshire South
- Newport North
- Newport East
- Newport West
- Torfaen North
- Torfaen South

It is known that GP practices in the same cluster give similar advice on which hospital their patients should attend. Caerphilly clusters and Blaenau Gwent West are advised to send their patients to Ysbyty Ystrad Fawr. Newport clusters, Monmouthshire South and Torfaen South are advised to send their patients to Royal Gwent. And Monmouthshire North, Blaenau Gwent East, and Torfaen North are advised to send their patients to Neville Hall. This is reflected is Figure 7.14, where GP practices from each cluster are overlayed onto a map where LSOAs are coloured by their closest hospital.

In addition to proximity, it is known that different hospitals specialise in different ailments. Therefore a patient may still need to be admitted to a hospital far away from their home, however this won't be common.

In order to account for this, the following probabilistic adjustment is proposed: each county c has three associated probabilities $a_{(\text{RG},c)}$, $a_{(\text{NH},c)}$, and $a_{(\text{YYF},c)}$; such that $a_{(\text{RG},c)}$ is the probability of a patient from county c going to Royal Gwent for a general ward admission, $a_{(\text{NH},c)}$ is the probability of a patient from county c going to

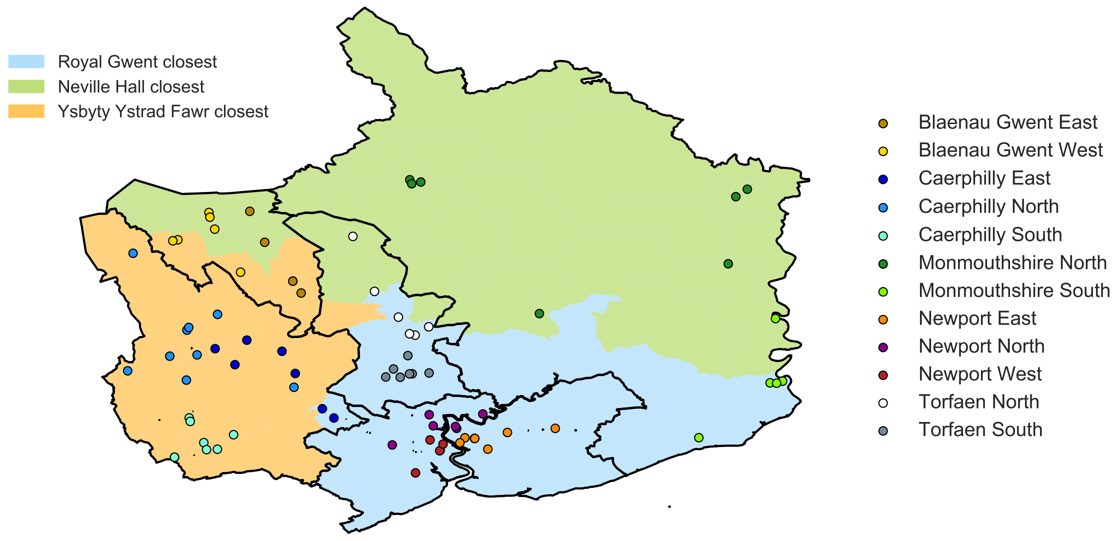


Figure 7.14: Overlay of locations of GP practices in the 12 clusters, on a map where LSOAs are coloured by their closest hospital.

Neville Hall for a general ward admission, and $a_{(\text{YYF},c)}$ is the probability of a patient from county c going to Ysbyty Ystrad Fawr for a general ward admission. Therefore $a_{(\text{RG},c)} + a_{(\text{NH},c)} + a_{(\text{YYF},c)} = 1$, for each county c. Now the parameters derived from the study cohort (a subset of Newport) can be adjusted by multiplying by the ratio of the probability of entering hospital h from county c, with the corresponding probability for Newport. For example the arrival rate to Neville Hall Hospital for patients from Caerphilly must be multiplied by $\frac{a_{(\text{NH,Caerphilly})}}{a_{(\text{NH,Newport})}}$. That is patients in from Caerphilly are $\frac{a_{(\text{NH,Caerphilly})}}{a_{(\text{NH,Newport})}}$ times more likely to be admitted to Neville Hall than patients from Newport (that is the study cohort).

In order to obtain $a_{(\text{RG},c)}$, $a_{(\text{NH},c)}$, and $a_{(\text{YYF},c)}$, the following steps are taken, making use of the Google Maps API:

1. Obtain the distances $d_{\text{RG},l}$, $d_{\text{NH},l}$, and $d_{\text{YYF},l}$ from the centre (centroid) of each LSOA l, to each hospital RG, NH and YYF.

2. Normalise the distances using an weighted inverse power normalisation method, weighted by number of beds at each hospital, to transform the distances to probabilities $a_{\text{RG},l}$, $a_{\text{RG},l}$, and $a_{\text{RG},l}$.

3. Aggregate the probabilities over each county, weighted by the population of people over 60 years of age, to obtain the probabilities $a_{(\text{RG},c)}$, $a_{(\text{NH},c)}$, and $a_{(\text{YYF},c)}$.

The weighted inverse power normalisation used in step 2 is shown in Equation 7.13, with parameter n. The d's represent the distances, and the w's represent the hospital weights. The hopsital weights are taken as the number of beds available at each hospital, thus $w_{\text{RG}} = 774$, $w_{\text{NH}} = 499$, and $w_{\text{YYF}} = 269$.

$$a_k = \frac{w_k d_k^{-n}}{\sum_j w_j d_j^{-n}} \tag{7.13}$$

This is more appropriate in this case than the usual Softmax normalisation, $a_k = \frac{e^{-d_k/w_k}}{\sum_i e^{-d_i/w_i}}$, which is very sensitive to scale. Another advantage with the inverse power normalisation is that it is parametrised by the power n. Using Scipy's `scipy.optimize.minimize` function [76], a parameter n can be chosen that yields a Newport vector closest to the observed values. This function minimises the mean absolute error between the predicted vector $\left(a_{(\text{RG},c)}, a_{(\text{NH},c)}, a_{(\text{YYF},c)}\right)$ and the corresponding observed values, for Newport.

Figure 7.15 shows how the mean absolute error between the fitted and observed vectors for the Newport parameters is affected by the choice of n. The optimal value of n using the minimisation algorithm was $n = 2.30896$. Figure 7.16 shows the obtained distances, probabilities, and aggregated probabilities for each hospital.

The external arrival rates into these three hospitals h for each county c will be adjusted by multiplying by a factor $\frac{a_{(h,c)}}{a_{(h,\text{Newport})}}$, that corresponds to how much more likely a patient is to attend hospital h from county c than from Newport (that is the study cohort). Table 7.8 summarise the calculated probabilities $a_{(\text{RG},c)}$, $a_{(\text{NH},c)}$, and $a_{(\text{YYF},c)}$ for each county.

These values transform the arrival rates and transition matrices in the following ways:

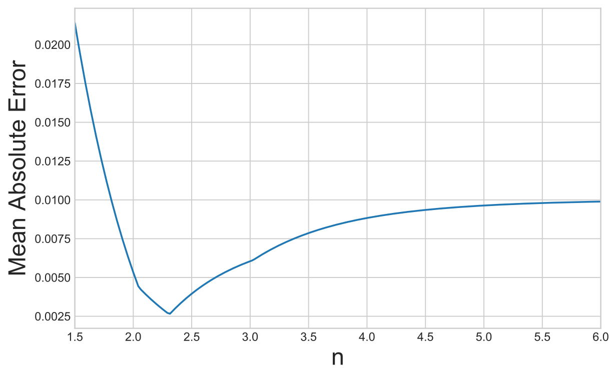


Figure 7.15: Effect of the choice of the parameter n of the normalisation method on the mean absolute error between the fitted and observed vectors for the Newport parameters.

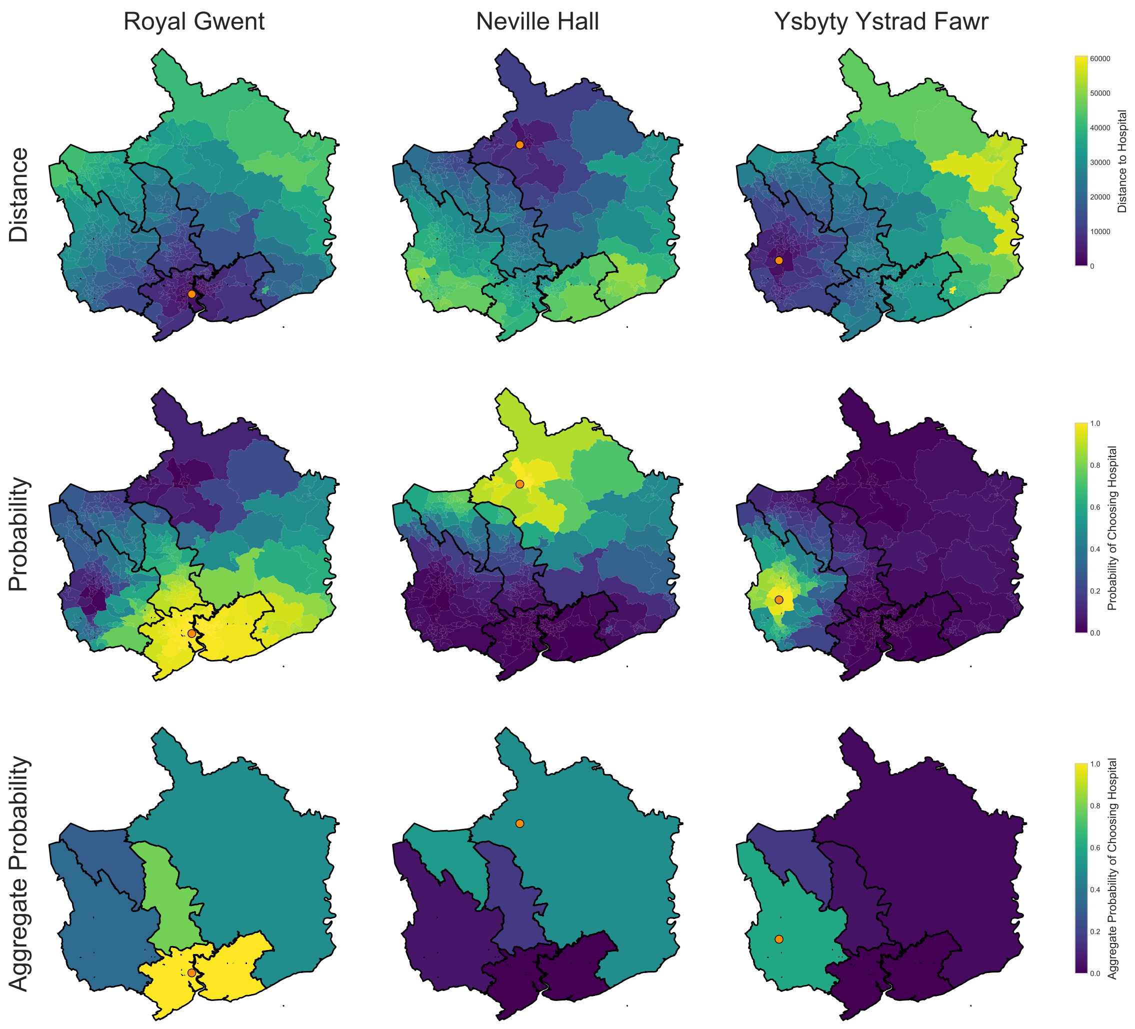


Figure 7.16: Geographic visualisation of the adjustment method.

	$a_{(\text{RG},c)}$	$a_{(\text{NH},c)}$	$a_{(\text{YYF},c)}$
Blaenau Gwent	0.295366	0.534659	0.169976
Caerphilly	0.346471	0.061861	0.591669
Monmouthshire	0.481640	0.482888	0.035472
Newport	0.984955	0.008104	0.006941
Torfaen	0.775453	0.168336	0.056211

Table 7.8: Table of the probability vector $(a_{(\text{RG},c)}, a_{(\text{NH},c)}, a_{(\text{YYF},c)})$ for each county.

1. For facilities f, Royal Gwent, Neville Hall, and Ysbyty Ystrad Fawr, the arrival rates are multiplied by the ratio $\frac{a_{(h,c)}}{a_{(h,\text{Newport})}}$ for each county c.
2. For each row in the transition matrices, the 2nd, 3rd and 4th entries correspond to the probability of being transferred to each hospital. These are replaced by $a_{(\text{RG},c)} \times q$, $a_{(\text{NH},c)} \times q$, and $a_{(\text{YYF},c)} \times q$ respectively, where q is the probability of being transferred to any hospital, that is the sum of the 2nd, 3rd and 4th entries of that row.

7.7 Model validation

In order to validate the model developed over the last few sections, the model is run for a simulated population with the size and characteristics of the study cohort. This simulation is run for 1 year, with a year's warm up time and a year's cool down time, over 100 trials. To confirm that the model is reflecting reality, some key performance indicators should correspond to the equivalent results observed in the data. Two indicators were used:

1. The total number of visits at each facility. This validates that the arrival rates and transition matrices reflect reality.
2. The total number of contact days at each facility. This validates that the interaction of service rates and demand reflect reality.

Figures 7.17 and 7.18 show histograms of the results obtained from 100 trials of the simulation run, with the orange vertical lines representing the observed value

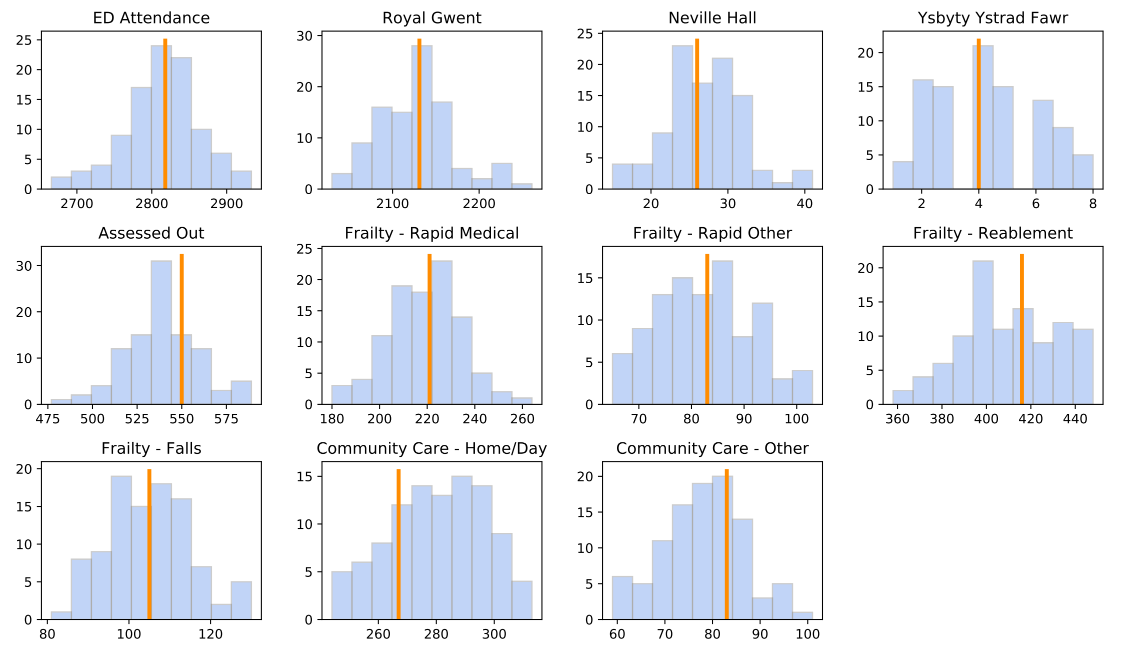


Figure 7.17: Validation run results for the demand on each facility.

from the data. These visualisations confirm the model reflects reality for the study cohort.

Note that, as discussed in Section 7.5, the service times for the three hospitals were amalgamated, and so service times for all three hospitals were sampled from the same distribution. This explains why the observed number of contact days for Ysbyty Ystrad Fawr is much larger than that simulated. During the last year of the observations, that which parameters were drawn, only four visits to Ysbyty Ystrad Fawr were recorded, one of which had a length of stay of 111 days. Therefore, with only four data points and one being an extreme value, there is not enough data to compare fairly here.

7.8 Workforce assumptions

Once simulations are run (the results of which can be found in Section 7.9), workforce requirements can be estimated from the results using the following assumptions:

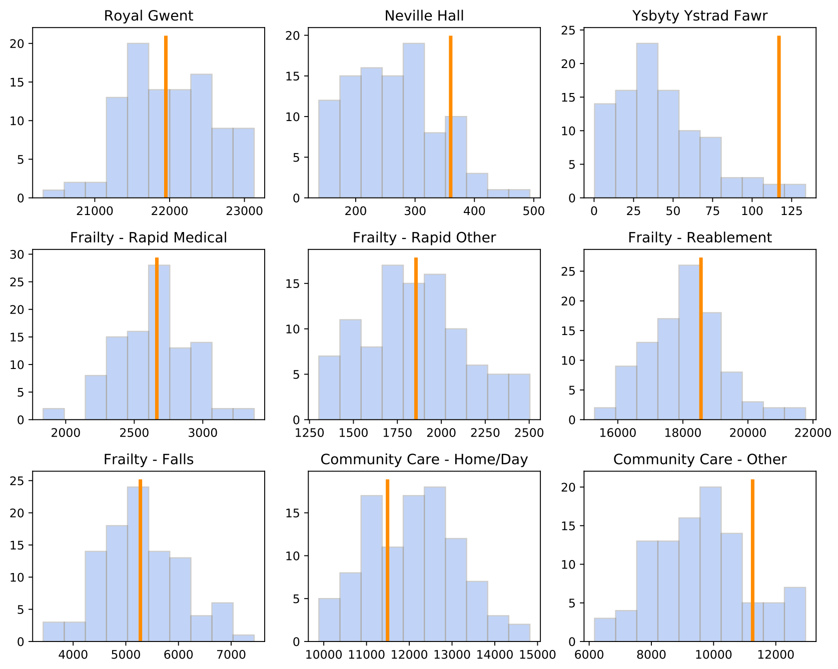


Figure 7.18: Validation run results for the contact days at each facility.

For general ward admissions there are two types of staff, band 5 nurses costing £30,339, and band 2 healthcare support workers costing £20,053. For each band 5 nurse there is one band 2 healthcare support worker. The numbers of staff are calculated as follows:

- Each day staff (nurse / healthcare support worker) is responsible for 7 beds. Each night staff is responsible for 11 beds.
- If $\bar{x}$ is the average number of contact days per year, then $b = \lceil \frac{\bar{x}}{365} \rceil$ is the number of beds required per day.
- Then the number of day staff per day is $n_d = \frac{b}{7}$ and the number of night staff per day is $n_n = \frac{b}{11}$.
- Assuming a day shift is 12 hours and a night shift is 12 hours, then the number of day staff hours required per week (plus 26.8% uplift to account for illnesses and absences) is $h_d = n_d \times 12 \times 7 \times 1.268$, and similarly the number of night staff hours per week is $h_n = n_n \times 12 \times 7 \times 1.268$.
- Finally the number of whole time equivalents required per week is $\frac{h_d+h_n}{37.5}$.

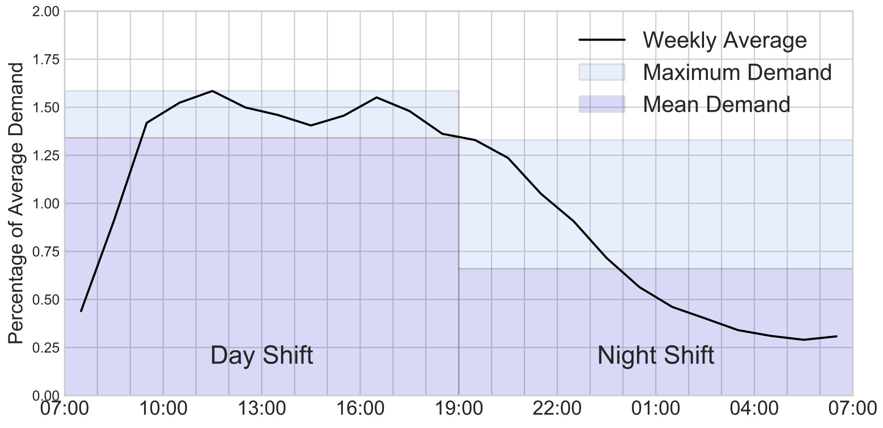


Figure 7.19: Total hourly demand at the A&E department at Royal Gwent for 2015, including two extreme approximations to this demand (maximum and mean) for day and night shifts.

It is known that demand at ED is not uniform throughout the day. However the simulation model built here does assume that, as the smallest level of granularity of the input data is days. In order to estimate the number of ED nurses required, this needs to be adjusted for. Figure 7.19 shows the total hourly demand at the A&E department at Royal Gwent for 2015.

Two 12 hour shifts are considered, the day shift that runs from 7am to 7pm and the night shift that runs from 7pm to 7am. As the exact nurse scheduling method cannot be determined at this level of granularity, two methods of adjusting for this hourly variation are considered, both are illustrated in Figure 7.19:

1. **Mean**: for each shift multiply the observed demand from the model by the ratio of the mean historical demand for that shift to the mean historical demand for the day.

2. **Max**: for each shift multiply the observed demand from the model by the ratio of the maximum historical demand for that shift to the mean historical demand for the day.

Now for ED nurses, whichever adjustment method used, we find the weekly number of nurses:

- Each nurse can treat 15 patients per 12 hours.
- Let $f_s(x)$ be the transform used to adjust the demand per shift s.
- If $\bar{v}$ is the average number of visits to ED per 24 hours, then $w_s = \left\lceil \frac{f_s(\bar{v}/2)}{15} \right\rceil$ is the number of required nurses on shift s per day.
- The number of nursing hours required per week (on shift s), with 26.8% uplift, is $h_s = w \times 12 \times 7 \times 1.268$.
- The total number of whole time equivalents required per week is then $h_s/37.5$ summed over both day and night shifts.

For Frailty services precise workforce data, such as contact times and which type of staff are required are not recorded, due to the more personalised nature of the services. The role of the care worker who has first contact with a patient for each frailty episode was given for a sample of observations in the data set. However, after the first contact no more staffing data is recorded. Thus for each of the four priorities, only workforce requirements for the initial contact can be predicted.

The proportions of roles of the initial contact over the four Frailty priorities are shown in Figure 7.20. This highlights the differences in the nature of the care of the four priorities, with Falls and Reablement requiring more occupational and physiotherapists, while Rapid services require more nursing care.

The four Frailty priorities are run by the same Frailty service, thus share staff. Therefore, for each of the eight roles concerned:

- For each priority multiply the total yearly number of visits by the role:priority proportion (shown in Figure 7.20), to obtain the yearly number of initial contacts required for that role for that priority.
- Sum this yearly number of initial contacts across the four priorities, to obtain the yearly number of contacts required for that role.

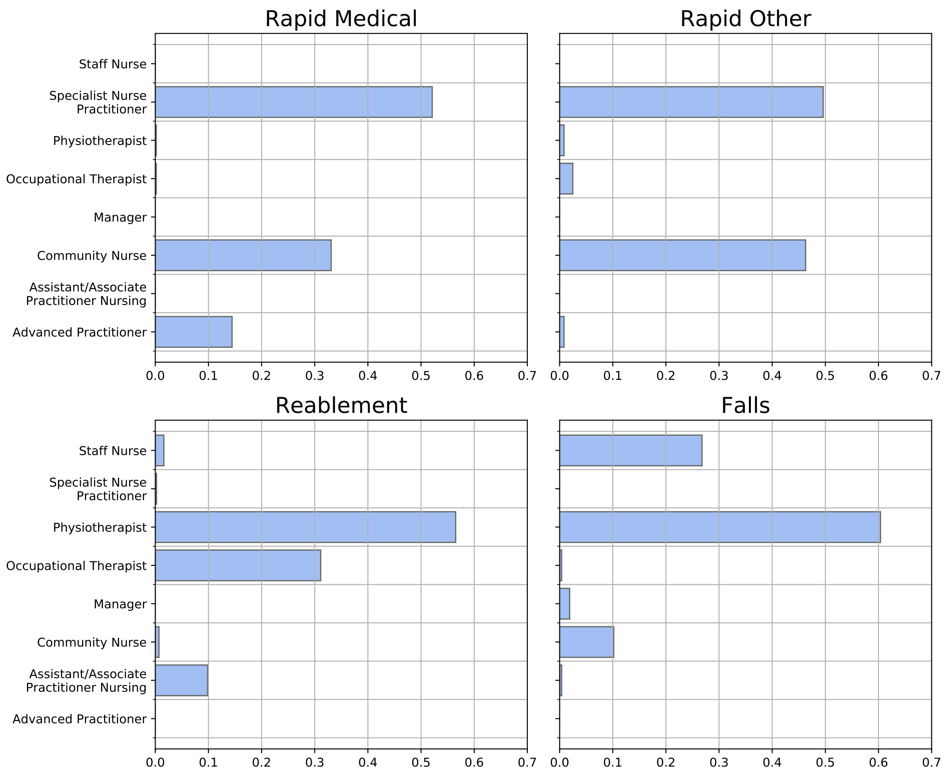


Figure 7.20: Proportions of the initial contacts taken up by different Frailty workers' roles, for each of the four Frailty priorities.

7.9 Results of the experiments

Three ordered scenarios were run:

- **Scenario 1 (Nowhere)**: Plans were not offered anywhere, that is $p_{(a,c)} = 0$ for all c. This reflects the situation before plans were introduced.
- **Scenario 2 (Newport)**: Plans are only offered in Newport, that is $p_{(a,c)} = 0$ except for $c =$ Newport. This reflects the current situation of plans only being on offer to residents of Newport.
- **Scenario 3 (Everywhere)**: Plans are offered in all counties of Gwent. This would reflect the intended future in which all GP practices in Gwent offer the plan.

Similarly to the validation runs, the simulations were run for 1 year, with a year's warm up time and a year's cool down time, over 100 trials. Total contact days were recorded at each facility. Boxplots of the results for the three scenarios, over 100

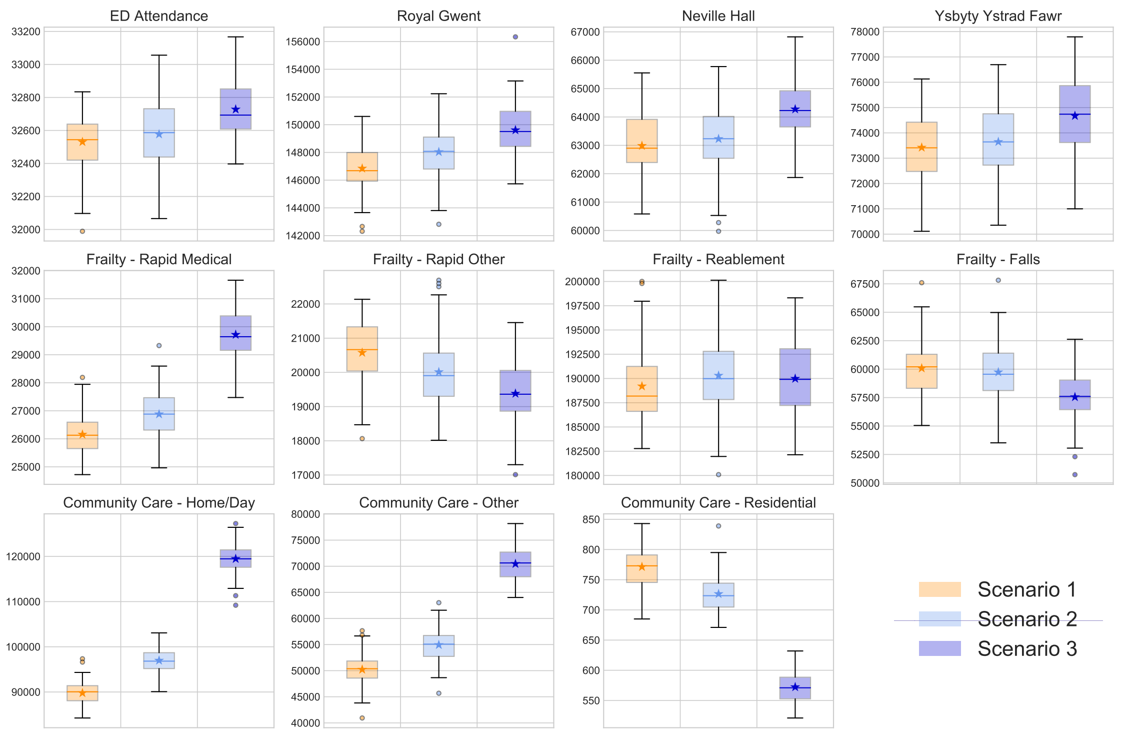


Figure 7.21: Yearly contact days for each facility, over the three scenarios (ED Attendance and Community Care - Residential show only numbers of visits).

trials, are shown in Figure 7.21.

The mean number of contact days (or visits for ED attendances and Residential care) per year, for each scenario, is given in Table 7.9. An ANOVA was conducted, testing whether the observed increases or decreases in demand are significant or imply down to chance. The null hypothesis H_0 is that there is no difference in the mean contact days (or visits) of each scenario, and the alternative hypothesis is that at least one of the scenarios has a significantly different mean. The results show that only Frailty - Reablement is unaffected by the presence of SWPs.

There are five major conclusions that can be draw from these results:

1. The adoption of SWPs will cause slight increases in demand at ED, and at general wards at all three hospitals.

2. The adoption of SWPs will cause a significant increase in demand at Rapid Medical Frailty services. This is offset however by a fall in demand at Rapid Other and Falls Frailty services.

	Scenario 1	Scenario 2	Scenario 3	p-Value	% Change
ED Attendances	32530.55	32577.00	32727.79	4.5×10^{-15}	+0.61%
Royal Gwent	146845.83	148034.61	149610.50	2.2×10^{-24}	+1.88%
Neville Hall	62986.25	63227.79	64273.26	9.9×10^{-16}	+2.04%
Ysbyty Ystrad Fawr	73422.42	73645.56	74674.49	4.3×10^{-10}	+1.71%
Frailty - Rapid Medical	26155.07	26879.68	29716.90	5.8×10^{-100}	+13.62%
Frailty - Rapid Other	20576.76	20010.73	19381.30	2.8×10^{-16}	-5.81%
Frailty - Reablement	189200.02	190301.36	189985.65	0.124 68	+0.42%
Frailty - Falls	60081.20	59734.90	57551.11	4.0×10^{-14}	-4.21%
Community Care - Home/Day	89775.52	96972.23	119472.67	2.4×10^{-197}	+33.08%
Community Care - Other	50218.44	54943.27	70468.13	1.4×10^{-145}	+40.32%
Community Care - Residential	771.34	726.53	572.27	5.1×10^{-149}	-25.81%

Table 7.9: Mean number of contact days (or visits for ED attendances and Residential care) per year, for each scenario, with reported p-Value of the ANOVA test.

3. Reablement Frailty services are the only facility to be unaffected by SWPs.
4. The adoption of SWPs will cause large increases in demand at Day, Home and Other Community Care services.
5. The adoption of SWPs will case large preventions of admittances to Residential Community Care services.

Following the workforce assumptions discussed in Section 7.8, the expected weekly number of WTE ED and general ward nurses and healthcare support workers can be found for each scenario with their respective costs, as well as the number of initial contacts carried out by frailty services staff. Tables 7.10, 7.11 and 7.12 summarise the results. ANOVA tests were undertaken to determine is the increase in workforce requirements were significant. The null hypothesis H_0 is that the required numbers of nurses is equal across all three scenarios, and the alternative hypothesis is that at least one scenario displays an increase in the expected number of weekly nurses required. The p-Values are reported in the tables, and we conclude that the increases or decreases observed from the simulation models are significant for all general ward and ED nurses. For ED nurses, the increase in required nurses from the Scenario 1 to Scenario 3 results in an increase in cost of £7,755.52 (1.02%) using the max method of scheduling, and £33,607.26 (12.29%) using the mean scheduling method. For general wards, the increase in staff required (both nurses and healthcare support

	ED (max)	ED (mean)
Scenario 1	25.56 (£775,552.22)	9.03 (£274,028.45)
Scenario 2	25.59 (£776,413.94)	9.37 (£284,369.15)
Scenario 3	25.82 (£783,307.74)	10.14 (£307,635.71)
p-Value	0.000 47	9.6×10^{-9}

Table 7.10: Expected number of weekly WTE ED nurses for the three scenarios, with reported p-Values of the ANOVA tests.

	Royal Gwent	Neville Hall	Ysbyty Ystrad Fawr
Scenario 1	257.80 (£7,821,270.64)	110.75 (£3,359,909.31)	129.04 (£3,914,804.05)
Scenario 2	259.92 (£7,885,732.12)	111.16 (£3,372,488.25)	129.44 (£3,927,181.55)
Scenario 3	262.75 (£7,971,546.47)	113.03 (£3,429,227.79)	131.26 (£3,982,399.08)
p-Value	1.3×10^{-24}	2.1×10^{-15}	5.8×10^{-10}

Table 7.11: Expected number of weekly WTE general ward nurses for the three scenarios, along with their costs, with reported p-Values of the ANOVA tests.

workers) across the three hospitals from Scenario 1 to Scenario 3 corresponds to a cost of £477,011.28 (1.9%).

For Frailty services, a significant increase in the initial contacts carried out by Specialist Nurse Practitioners, Community Nurses and Advanced Practitioners is observed, but also a significant decrease in the required numbers of Staff Nurses and Managers (the latter having a reduction of less than one contact a year however). This is summarised in Table 7.13.

As was seen in Section 7.3, the population of Gwent is likely to increase by over 30% in the next 25 years. How does the effects seen in Figure 7.21 scale as the population grows? The simulation model was run for each year between 2015 and 2039 with the projected populations fed into the model parameters. The resulting

	Royal Gwent	Neville Hall	Ysbyty Ystrad Fawr
Scenario 1	257.80 (£5,169,581.73)	110.75 (£2,220,780.56)	129.04 (£2,587,546.25)
Scenario 2	259.92 (£5,212,188.48)	111.16 (£2,229,094.79)	129.44 (£2,595,727.34)
Scenario 3	262.75 (£5,268,908.71)	113.03 (£2,266,597.61)	131.26 (£2,632,224.16)
p-Value	1.3×10^{-24}	2.1×10^{-15}	5.8×10^{-10}

Table 7.12: Expected number of weekly WTE general ward health care support workers for the three scenarios, along with their costs, with reported p-Values of the ANOVA tests.

	Scenario 1	Scenario 2	Scenario 3	p-Value
Advanced Practitioner	336.05	341.29	363.34	1.2×10^{-89}
Assistant/Associate Practitioner Nursing	442.99	443.57	441.89	0.176 39
Community Nurse	1324.73	1329.21	1372.11	1.3×10^{-47}
Manager	23.19	23.10	22.54	4.1×10^{-12}
Occupational Therapist	1417.50	1419.06	1414.00	0.204 74
Physiotherapist	3271.70	3272.02	3245.55	5.9×10^{-6}
Specialist Nurse Practitioner	1641.44	1652.95	1727.71	6.3×10^{-68}
Staff Nurse	400.68	399.42	391.34	2.0×10^{-12}

Table 7.13: Expected number of yearly initial contacts carried out by frailty staff for the three scenarios, with reported p-Values of the ANOVA tests.

predicted demand at each facility for each year between 2015 and 2039 is shown in Figure 7.22.

Though these results do not show any scaling of effect with population increase, it does highlight important features on the scale of effects for each facility. The secondary care facilities show that the differences observed in each scenario due to the SWPs are substantially dwarfed by the natural increase in demand by population increase from year to year. Thus any positive or negative effects at these facilities are likely to go unnoticed in comparison with the natural changes in demand caused by population increase. Frailty services show moderate effect sizes compared to the effect of population growth, however it is in Community care services where large scale effects due to SWPs are seen. In fact by 2039 the number of yearly admissions to residential care in the scenario where SWPs are offered everywhere is roughly equal to the number of yearly admissions to residential care in 2015 in the scenario where there are no SWPs. The savings towards residential care admissions are large, even when the cumulative effect of the years is not considered.

An immigration-death model of residential care

The cumulative effect of residential care admissions can be estimated using a time-dependent immigration-death model. A representation of this Markovian immigration-death model is shown in Figure 7.23. The states of the continuous time Markov chain correspond to the current number of residents in Residential care. Thus sum-

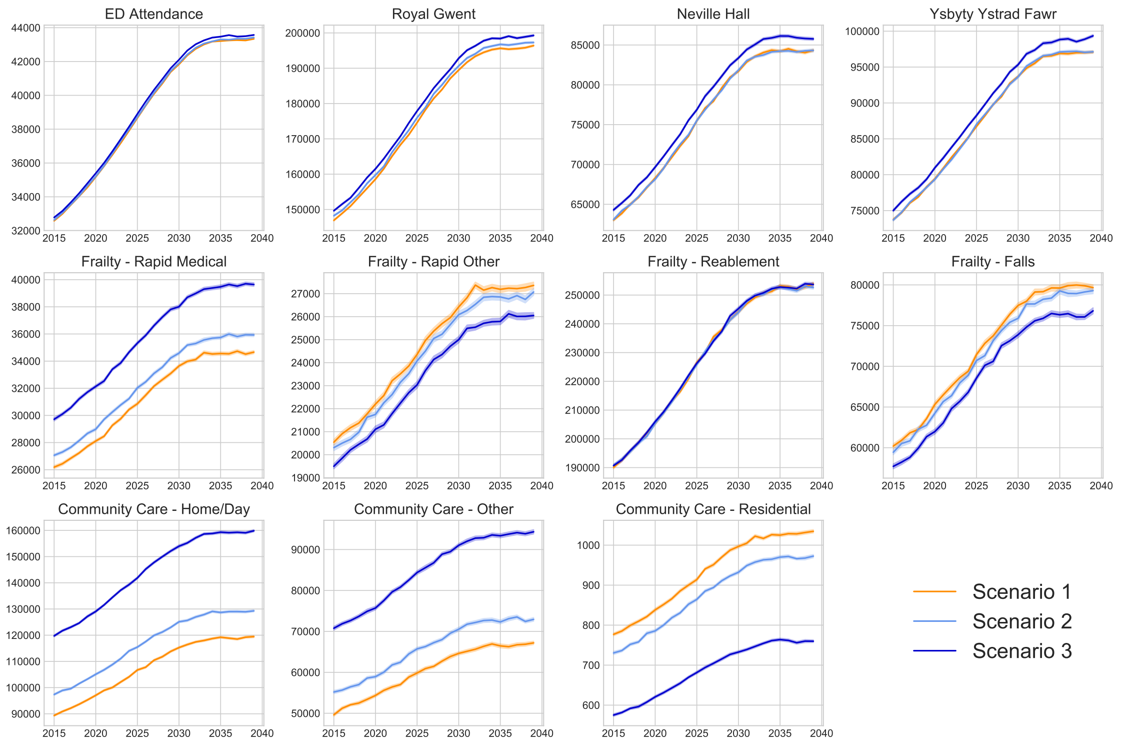


Figure 7.22: Projected yearly contact days for each facility, over the three scenarios (ED Attendance and Community Care - Residential show only numbers of visits), for the period 2015 - 2039.

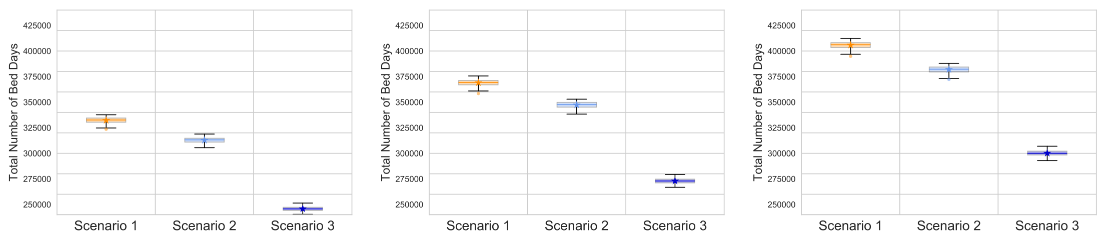
$$\begin{tikzcd}
(\mathbf{0}) \arrow[r, bend left, "\Lambda(t)"] & (\mathbf{1}) \arrow[l, bend left, "\mu"] \arrow[r, bend left, "\Lambda(t)"] & (\mathbf{2}) \arrow[l, bend left, "2\mu"] \arrow[r, bend left, "\Lambda(t)"] & \cdots \arrow[l, bend left, "3\mu"] \arrow[r, bend left, "\Lambda(t)"] & (\boldsymbol{n}) \arrow[l, bend left, "n\mu"] \arrow[r, bend left, "\Lambda(t)"] & \cdots \arrow[l, bend left, "(n+1)\mu"]
\end{tikzcd}$$

Figure 7.23: Transition diagram of the time dependant immigration-death process used to model Residential Community Care occupancy.

ming up the number of residents multiplied by the time since the last event will give the total number of bed days used over the simulation period. The same three scenarios are run.

Patients are admitted at a time-dependant rate $\Lambda(t)$, the 'immigration rate' of the model. For each scenario this corresponds the the yearly admissions calculated from the simulation model, shown in Figure 7.22.

Patients leave residential care when they pass away. Thus the 'death rate' of the model, μ, is the death rate of the patients at Residential care. ONS data is used to obtain this rate, under the assumption that patients over 60 years of age in Gwent residential care homes have the same death rate as all people over 65 in



(a) Death rate -10% (b) Death rate +0% (c) Death rate +10%

Figure 7.24: Results of the immigration-death model, with expected total number of bed days used in 25 years shown for each scenario.

England and Wales. From mid 2015 estimates, there are 10336345 people over the age of 65 living in England and Wales, and there were 449409 deaths of people over the age of 65 in England and Wales. Thus there is a death rate of $\mu = 0.04348$. However, this death rate is an estimate only: it is not known whether residential care residents have a higher death rate than the rest of the population as they are iller than most other people, or if they have a lower death rate as they are being cared for professionally. Therefore two more scenarios are run where the death rate is increased and decreased by 10%.

The model is run for 25 years (from 2015 to 3039, for which the population projections are estimated). As we are only interested in the gains made from rolling out SWPs, thus the differences in occupancy, then the model begins from an empty system, that is state (0). The system is simulated over 100 trials, and Figure 7.24 show the total number of bed days over a 25 year period used for each scenario.

The model shows a large decrease in bed days as more SWPs are given out. Table 7.14 summarises the total number of residential care bed days used for each scenario. A one-way ANOVA is carried out, yields a p-values close to zero in each case. Thus the improvements seen in the plots are significant. Scenario 3 exhibits a 26.0% decrease in the amount of Residential care bed days used over 25 years compared to the Scenario 1 (regardless of the death rate used).

In order to estimate staffing levels and costs of residential care it is assumed that the nurse to bed ratio is 1:12, and that a nurse of band 2 is required, costing £20,053

	Scenario 1	Scenario 2	Scenario 3	p-Value
-10%	332208.81	312808.76	245728.90	0.0
+0%	368996.16	347271.07	272940.23	0.0
+10%	405744.54	381965.06	300107.83	0.0

Table 7.14: Mean number of residential care bed days for each scenario, with reported p-Value of the ANOVA test.

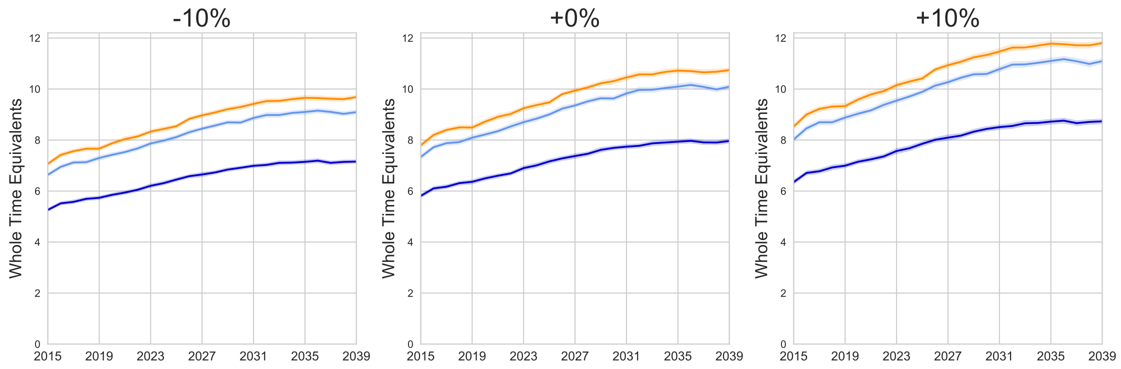


Figure 7.25: Projected yearly staffing levels at Community Care - Residential, for the period 2015 - 2039.

a year per whole time equivalent nurse. Figure 7.25 shows the yearly predicted staffing levels for each scenario for the years 2015 to 2039. This is summarised, along with the associated costs, in Table 7.15. These results confirm that the the savings made by SWPs at residential care are cumulative over time. For 2015, the savings in WTEs made by rolling out SWPs across Gwent (that is the difference in WTEs needed for Scenario 1 to the Scenario 3) is 1.98, or a cost of £39,774.920. For 2039, these savings increase to 2.78, or a cost of £55,770.330. That is, the benefits seen in 2015 by implementing SWPs increases 37.03% by 2039. Cumulatively, this corresponds to a total saving over the 25 years of £1,247,784.10.

7.10 Summary of results

In Section 7.1, three questions were set out to be addressed in this analysis. In subsequent sections, through data analysis and modelling techniques, some insights have been found. Here those insights are summarised.

	Scenario 1	Scenario 2	Scenario 3
2015	7.790 (£156,251.86)	7.340 (£147,091.29)	5.810 (£116,476.94)
2019	8.490 (£170,301.24)	8.090 (£162,279.81)	6.360 (£127,583.54)
2023	9.250 (£185,442.29)	8.700 (£174,478.08)	6.900 (£138,357.89)
2027	9.940 (£199,301.82)	9.360 (£187,625.64)	7.370 (£147,850.72)
2031	10.45 (£209,601.53)	9.830 (£197,071.00)	7.740 (£155,207.65)
2035	10.72 (£215,012.44)	10.09 (£202,434.45)	7.940 (£159,147.17)
2039	10.74 (£215,439.61)	10.09 (£202,339.52)	7.960 (£159,669.28)

Table 7.15: Summary of the predicted staffing levels and costs of residential care, for the case when death rate +0%.

i. What effects are Stay Well Plans having on how the population of older people in Newport are accessing the healthcare system?

It has been observed that patients that received a SWP, compared to similar patients who did not receive a SWP, were:

- More likely to be referred home as opposed to staying in the healthcare system. That is, they were less likely to be referred to a general ward after an ED visit, less likely to be referred to local authority community care services after a hospital stay, and less likely to require local authority community care services after a reablement frailty episode.

- Using the healthcare system less on each visit. That is lengths of stays in general wards were reduced, and lengths of frailty episodes for the rapid other, reablement, and falls priorities were reduced.

ii. What would be the effect on demand and workforce needs of the healthcare system if the plans were to be rolled out to all five counties served by Aneurin Bevan University Health Board?

The results of the simulations have enabled prediction of the demand and workforce requirements at each investigated healthcare facility. The simulation results show that the roll-out of SWPs across all five counties:

- Demand at secondary care facilities is expected to increase slightly. Attendances at ED for persons over 60 years of age is expected to increase by 0.61%; demand at Royal Gwent general wards are expected to increase by 1.88%, demand at Neville Hall by 2.04%, and demand at Ysbyty Ystrad Fawr by 1.71%.
- The increase in demand at ED may result in an increase in nurse requirements between 12.29% with extremely unconservative scheduling, and 1.02% with extremely cautious scheduling. That is absolute increase in costs of between £33,607.26 and £7,755.52 .
- The increase in demand at general wards may result in staff increase (both nurses and healthcare support workers) of 1.9%, or an absolute increase in costs of £477,011.28 .
- Demand at frailty services is expected to be redistributed amongst the four priorities: demand at Rapid Medical is expected to increase by 13.62%, demand at Rapid Other is expected to decrease by 5.81%, demand at Falls to decrease by 4.21%, while Reablement is unaffected by the plans.
- This redistribution of demand at Frailty services will cause very slight increases in the number of initial contacts carried out by Advanced Practitioners, Community Nurses and Specialist Nurse Practitioners; and very slight decreases in the number of initial contacts carried out by Managers, Physiotherapists and

Staff Nurses.

- Demand and preventative community care services, Home/Day and Other, is expected to have large increases in demand, 30.08% and 40.32% respectively.
- Demand at residential community care services is expected to decrease by a large amount, 25.81%.

iii. How do these effects, if any, scale with projected population growth?

The population of people over 60 years of age living in Gwent is projected to increase over the next 25 years, in total by around 25%. Regardless of the presence of SWPs, this population growth will cause demand at all facilities investigated in this analysis to increase. Modelling this natural population increase in the presence and absence of SWPs gives insights to the scale of SWP benefits in comparison to this natural growth.

- The increases in demand at secondary care due to the roll-out of SWPs is dwarfed in comparison to the natural increase in demand due to population growth. The negative effects of SWPs are inconsequential here over the span of more than a year.
- In comparison to the increase in demand from population growth, the decreases in demand due to SWPs at Rapid Other and Falls frailty services are small. However the increase in demand due to SWPs at Rapid Medical services do seem meaningful.
- The increases in demand due to SWPs at preventative community care services (Home/Day and Other) are large in comparison to the increases in demand due to population growth. In fact the demand in 2039 with no SWPs is less than the demand in 2015 with SWPs everywhere.

- The decreases in demand due to SWPs are residential community care is large in comparison to the increases in demand due to population growth. Again here, the demand in 2015 with no SWPs is greater than the demand in 2039 with SWPs everywhere.
- The affect of residential care is cumulative over the 25 years, as residents stay longer than one year. Over the 25 years it is expected that there will be a 26.0% decrease in the amount of residential care bed days used if the plan is rolled out across Gwent. This corresponds to a total saving of £1,247,784.10 over 25 years.

The code used to produce the analysis of this chapter is partially available at [127] with DOI `10.5281/zenodo.1420403`, however some lines of code have been censored and the data is unavailable due to its sensitive nature.

Chapter 8

Conclusions

This chapter serves to summarise the work and contributions of this thesis. Each chapter contains a detailed summary section, and so the summary here will be brief.

8.1 Research summary

Chapter 1 introduced the initial research task of this thesis, to evaluate the effect of Stay Well Plans on the population of older people in Gwent. The methodology chosen, simulation and queueing networks, gave rise to further interesting research questions, namely the question of conducting reproducible discrete event simulations, and the investigation of the phenomenon of deadlock.

Chapters 2, 5, and 6 discussed deadlock in open restricted queueing networks in great detail:

- Two types of deadlock were defined, transient and absolute deadlock;
- Possible deadlock configurations were enumerated in complete queueing networks;
- A dynamic graph theoretical construction, the state digraph, was defined, and its properties are used to detect deadlock in simulations of restricted queueing

networks;

- Markov models of three simple queueing networks were defined and used to calculate the expected time to deadlock from an empty system;
- Investigations into how system parameters such as arrival and service rates, transition probabilities, queueing capacity and number of servers effect the time to deadlock were carried out;
- A deadlock resolution procedure, and a corresponding graph theoretic construction, were defined and discussed, with results on constrained unblockages presented;
- A demonstrative Markov decision process was formulated to explore whether some optimal deadlock resolution policy could improve performance.

Chapters 3 and 4 present the case for a new discrete event simulation framework that enables reproducible discrete event simulation models. A new Python library, Ciw, was developed to meet this need. These chapters journal its development, using best practices in research software development, and outlines its usage and features.

Finally in Chapter 7 the initial research question is answered. A simulation model of a healthcare system in Gwent, consisting of 12 health facilities, is built and parametrised from data. Data from a small geographical area was obtained, and so a novel approach to extrapolating this data across a large geographical area was utilised, considering population sizes, deprivation levels, and proximity to hospitals. This model produced expected demand at each facility, and further workforce models were then used to obtain workforce needs. Three scenarios were run of different levels of SWP prevalence, which gave insights into their effect. The broad observed effects were:

- Patients more likely to be referred home than to other healthcare facilities;
- Slight increases in demand at secondary care facilities;

- A redistribution of demand amongst frailty services;
- Large increases in demand at home and other community care services;
- Large decrease in demand at residential care homes.

8.2 Contributions

This thesis has made novel contributions across three themes. The phenomenon of deadlock in open restricted queueing networks has been investigated in depth. Two graph theoretical constructions have been given, the first is used to detect deadlock, and the second is used analyse deadlock resolution. Furthermore Markov models of deadlock have been built in order to calculate the expected time to deadlock, a statistic not looked at before. These explorations have enabled a discussion on deadlock both in reality and as a byproduct of mathematical models, and as such will be of use to modellers of systems with feedback loops.

Complementing this, an internationally used piece of open-source software has been developed with the aim of enabling reproducible discrete event simulations in Python. This was crucial in facilitating the research into deadlocking systems. It was also used to model a healthcare system in Gwent, with the aim of evaluating the effect of SWPs on the demand and workforce needs of healthcare facilities across secondary care, frailty services, and community care services. Real valuable insights were achieved by the modelling, glimpsing at an optimistic future in which SWPs, although slightly increasing demand at some facilities, aid in keeping older people out of residential care for longer.

8.3 Future research directions

Each part of this thesis has given rise to further interesting questions and research directions that, although not in the scope of the current work, would improve or compliment it. Let's take each theme in turn: deadlock, reproducible simulations, and evaluating SWPs.

Further work - deadlock

In Chapter 2 deadlock configurations were enumerated for complete queueing networks. In reality not all systems are complete. More often than not some routing possibilities will be closed off to some customers, that is the transition probability will be zero. One research question that arises is, if a subset of these routing possibilities are closed, how many deadlock configurations are now impossible to reach? That is enumerating deadlock configurations for non-complete queueing networks. This is a question closely related to enumerating and partitioning the number of cycles in an arbitrary digraph.

In Chapter 5 a list of all possible queueing networks with two of fewer nodes was given. Markov models were built for all except the two node multi-server system with self loops, and further research is needed to build a Markov model of this system. In the systems modelled in that chapter, customers only have one potential destination, and so customers may only get blocked from moving to one destination. In the two node network with self loops, but with single servers, although customers have two destinations, a blockage to the same node immediately results in deadlock. In all these cases, the unblocking mechanism is simple, as there is only ever one option of which node a customer joins when unblocked. However in the multi-server system with self loops, there are two destination nodes to which a customer may join when unblocked. Therefore, any representations of any states with blocked customers also need to hold information about these customers' destination nodes.

In addition to this, the order in which customers become blocked is important. In the models built in this thesis, when space become available at a node there is only one other node from which a blocked customer can become unblocked, however in the multi-server system of interest a node that has space available must accept the customer that has been blocked longest to that node. Therefore all states with blocked customers are also required to record the order in which the customers become blocked. Combining the two requirements above, it is clear that as the number of servers increases, the size of the state space for this queueing network quickly grows combinatorially. Therefore an entirely different state representation is required to model the two node multi-server system with self loops.

This chapter also gave large consideration to the expected time to deadlock, ω of a system from an empty state. This has only been calculated numerically, replying on software to invert matrices of numbers. Thus a natural extension to this work would be to find explicit equations for ω in terms of the parameters of the system. Such a results would also serve to complete some other unanswered questions of this chapter: they could be used to prove conjectures on the behaviour of the functions $\omega(\mu)$ and $\omega(\mu_1, \mu_2)$; and they could aid in finding a tighter bound in Section 5.4.2.

Finally Chapter 6 concluded somewhat unsatisfactorily; the two proof-of-concept examples given for the MDP did not yield much of a better performance than using a random policy, querying the value of such an endeavour. In order to draw any real conclusions about the benefits of finding an optimal deadlock resolution policy, this needs to be tested on a much larger collection of examples with a wide variety of parameter sets. This would provide a sufficiently large enough data set to decide for certain if this technique yields any advantages over random policies, and would give opportunity to investigate which parameters and properties of queueing networks would make such an task worthwhile.

In addition both examples in this chapter were two node systems, thus there was no need to incorporate the choice of subset S of customers into the action set. Finding all feasible S, and thus the proposed action set of a system with more than two nodes, is a challenge, equivalent to finding all possible directed Eulerian subgraphs of any directed graph. This further research direction may only be lucrative if it is shown that for some queueing systems an optimal deadlock resolution policy performs substantially better than a random policy.

One final future research question from this chapter has been discussed in detail in the chapter itself, that is using model free reinforcement learning to find optimal deadlock resolution policies. This would abolish the need to find all deadlock configurations beforehand, and increase accuracy by exploring a simulation of the system itself, rather than a model built from a number of simulation runs. There would still be a requirement to find all possible action though, rendering any network containing more than two nodes problematic.

Further work - reproducible simulations

In Chapter 3 a Python library for conducting reproducible discrete event simulations was built, Ciw. In order for this to always meet this goal, continuous updates, development, maintenance, promotion and support is required. Maintenance and support are obligatory to ensure that models built using Ciw are still runnable and reproducible. Out of date software, software that does not work due to its other software requirements changing or being updated, or software with no support cannot be reused to recreate the same results as before. Thus rendering models irreproducible. Development of new features is also vital in order for the software to be as applicable to as many simulation projects as possible, encouraging the writing of reproducible simulation models. How recent an entry is to a software's changelog, or how recent its latest commit has been made, is often an indicator of how likely the software is to be picked up by new users. So, continued promotion of the software will extend its reach, complementing the continued updating of the software.

More specifically two areas in Chapter 3 highlighted where further development of the code could be taken. Runtime speeds was an area where Ciw could not compete realistically with the other simulation options that were tested. Although speed is not a priority for Ciw, it may be a priority for some users, inhibiting their uptake of the software. Complementing this, a better knot detection algorithm could be implemented, as opposed to the brute force implementation currently in place. This would improve the runtime of simulations where deadlock detection is in place.

Further work - evaluating SWPs

Many aspects of the model produced in Chapter 7 could be improved to increase accuracy or usefulness of results. A number of decisions or limitations of this work were driven by the availability and quality of data. The following lists aspects that were initially considered however due to data restrictions were omitted from the analysis:

- A better workforce model for frailty services, such that similar analyses to ED and General Wards could be undertaken; however due to difficulties in obtaining data only initial contacts could be investigated.
- A better workforce model for residential community care services; however due to the residential care homes not being part of ABUHB data could not be obtained and an approximate model was given instead.
- The modelled healthcare facility 'ED' in fact contains attendances at A&E (Accident and Emergency), MAU (Medical Assessment Unit) and SAU (Surgical Assessment Unit), and separating these could give more accurate workforce data; however the data set did not differentiate between these facilities, and so one overarching facility was modelled instead.
- Some aspects of the system were omitted from the model entirely: routing between hospitals, and CHC (Continuing health care). Routing between hospitals were not captured by the data set, and CHC services could not be modelled as a queueing system due to its omnipresent nature.

Adjustments to the model to include and overcome these limitations would increase the model accuracy, and may give further insights into the effect of SWPs on the

Gwent population.

Furthermore the ‘cost’ of increased longevity (lower rate of admittance to residential care) is not completely considered in this work. Many costs are indirectly considered by the model, that is if there are less patients being admitted to residential care, then there is more demand on other parts of the system. Further work here could include considering this effect of patients quality of life, by measuring QALYs (quality-adjusted life years) and SWPs’ effects on them.

Bibliography

[1] Mark Aberdour. Achieving quality in open-source software. *IEEE Software*, 24(1), 2007.

[2] I. Akyildiz. Product form approximations for queueing networks with multiple servers and blocking. *IEEE Transactions on Computers*, 38(1):99–114, 1989.

[3] S. Albin, J. Barrett, D. Ito, and J.E. Mueller. A queueing network analysis of a health center. *Queueing Systems*, 7(1):51–61, 1990.

[4] G. Allon, S. Deo, and W. Lin. The impact of size and occupancy of hospital on the extent of ambulance diversion: Theory and evidence. *Operations Research*, 61(3):544–562, 2013.

[5] C. Ancker Jr and A. Gafarian. Some queueing problems with balking and reneging I. *Operations Research*, 11(1):88–100, 1963.

[6] C. Ancker Jr and A. Gafarian. Some queueing problems with balking and reneging II. *Operations Research*, 11(6):928–937, 1963.

[7] Aneurin Bevan University Health Board. ABCi - aneurin bevan continuous improvement. Accessed: 2018-06-14.

[8] Aneurin Bevan Univserity Health Board. About our organisation. Accessed: 2018-06-12.

[9] Apache. Subversion. `https://subversion.apache.org/`.

[10] Atlassian. BitBucket. `https://bitbucket.org/`.

[11] SWM Au-Yeung, PG Harrison, and WJ Knottenbelt. A queueing network model of patient flow in an accident and emergency. In *Proceedings of 2006 European Simulation and Modelling Conference, August*, pages 60–67, 2006.

[12] B. Avi-Itzhak and M. Yadin. A sequence of two servers with no intermediate queue. *Management Science*, 11(5):553–564, 1965.

[13] J. Baber. *Queues in series with blocking.* PhD thesis, Cardiff University, 2008.

[14] Norman TJ Bailey. A study of queues and appointment systems in hospital out-patient departments, with special reference to waiting-times. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 185–199, 1952.

[15] Norman TJ Bailey. Queueing for medical care. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 3(3):137–145, 1954.

[16] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications.* Springer Science & Business Media, 2008.

[17] Forest Baskett, K Mani Chandy, Richard R Muntz, and Fernando G Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975.

[18] N. Batchelder. Coverage. `https://coverage.readthedocs.org/`, 2017.

[19] F. Belik. An efficient deadlock avoidance technique. *IEEE Transactions on Computers*, 39(7):882–888, 1990.

[20] P.C. Bell and R.M. O'Keefe. Visual interactive simulation—history, recent developments, and major issues. *Simulation*, 49(3):109–116, 1987.

[21] V. Belton and M.D. Elder. Decision support systems: Learning from visual interactive modelling. *Decision Support Systems*, 12(4-5):355–364, 1994.

[22] Fabien Benureau and Nicolas Rougier. Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. *arXiv preprint arXiv:1708.08205*, 2017.

[23] SC Brailsford, Paul Robert Harper, B Patel, and M Pitt. An analysis of the academic literature on simulation and modelling in health care. *Journal of Simulation*, 3(3):130–140, 2009.

[24] Kurt M Bretthauer, H Sebastian Heese, Hubert Pun, and Edwin Coe. Blocking in healthcare operations: a new heuristic and an application. *Production and Operations Management*, 20(3):375–391, 2011.

[25] P. Burke. The output of a queueing system. *Operations Research*, 4(6):699–704, 1956.

[26] Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *Journal of Discrete Algorithms*, 9(3):231 – 240, 2011.

[27] P. Cameron. *Combinatorics: topics, techniques, algorithms.* Cambridge University Press, 1994.

[28] S. Chacon and B. Straub. *Pro git.* Apress, 2014.

[29] J. Cheng. Task-wait-for graphs and their application to handling tasking deadlocks. In *Proceedings of the Conference on TRI-ADA'90*, pages 376–390. ACM, 1990.

[30] H. Cho, T. Kumaran, and R. Wysk. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(3):413–421, 1995.

[31] E. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3(2):67–78, 1971.

[32] S. Creemers and M. Lambrecht. Modelling a healthcare system as a queueing network: the case of a Belgian hospital. *Available at SSRN 1093618*, 2007.

[33] Tom Crick, Benjamin A Hall, Samin Ishtiaq, and Kenji Takeda. Share and enjoy: Publishing useful and usable scientific models. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 957–961. IEEE Computer Society, 2014.

[34] Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Ildikó Schlotter. Parameterized complexity of Eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014.

[35] Georgios Dagkakis and Cathal Heavey. A review of open source discrete event simulation software for operations research. *Journal of Simulation*, 10(3):193–206, 2016.

[36] Y. Dallery and Y. Frein. On decomposition methods for tandem queueing networks with blocking. *Operations Research*, 41(2):386–399, 1993.

[37] Arnoud M. de Bruin, A. C. van Rossum, M. C. Visser, and G. M. Koole. Modeling the emergency cardiac in-patient flow: an application of queuing theory. *Health Care Management Science*, 10(2):125–137, 2007.

[38] B. Deuermeyer, G. Curry, A. Duchowski, and S. Venkatesh. An automatic approach to deadlock detection and resolution in discrete simulation systems. *INFORMS Journal on Computing*, 9(2):195–205, 1997.

[39] R Devillers. Game interpretation of the deadlock avoidance problem. *Communications of the ACM*, 20(10):741–745, 1977.

[40] E. Dijkstra. The mathematics behind the Banker's algorithm. In *Selected Writings on Computing: A Personal Perspective*, pages 308–312. Springer, 1982.

[41] Ralph L Disney and Dieter König. Queueing networks: A survey of their random processes. *SIAM Review*, 27(3):335–403, 1985.

[42] Bharat T Doshi. Queueing systems with vacations—a survey. *Queueing Systems*, 1(1):29–66, 1986.

[43] A. Elmagarmid. A survey of distributed deadlock detection algorithms. *ACM Sigmod Record*, 15(3):37–45, 1986.

[44] Agner Krarup Erlang. The theory of probabilities and telephone conversations. *Nyt. Tidsskr. Mat. Ser. B*, 20:33–39, 1909.

[45] J. Ezpeleta, F. Tricas, F. García-Vallés, and J.M. Colom. A Banker's solution for deadlock avoidance in FMS with flexible routing and multiresource states. *IEEE Transactions on Robotics and Automation*, 18(4):621–625, 2002.

[46] FarmacologiaClinica. Charlson index. `http://farmacologiaclinica.info/scales/Charlson_Comorbidity/`.

[47] G Fink and M Bishop. Property-based testing; a new approach to testing for assurance. *Software Engineering Notes*, 22(4):74–80, 1997.

[48] M. Florian, M. Mahut, and N. Tremblay. Application of a simulation-based dynamic traffic assignment model. *European Journal of Operational Research*, 189(3):1381–1392, 2008.

[49] Samuel Fomundam and Jeffrey W Herrmann. A survey of queuing theory applications in healthcare. Technical report, 2007.

[50] Alan Gibbons. *Algorithmic graph theory.* Cambridge University Press, 1985.

[51] Jennifer Gillespie, Sally McClean, F FitzGibbons, Bryan Scotney, Frank Dobbs, and BJ Meenan. Do we need stochastic models for healthcare? the case of ICATS. *Journal of Simulation*, 8(4):293–303, 2014.

[52] Git-scm. Git. `https://git-scm.com/`.

[53] GitHub Inc. GitHub. `https://github.com/`.

[54] GitLab. GitLab. `https://about.gitlab.com/`.

[55] William J Gordon and Gordon F Newell. Closed queuing systems with exponential servers. *Operations Research*, 15(2):254–265, 1967.

[56] W.J. Gordon and G.F. Newell. Cyclic queuing systems with restricted length queues. *Operations Research*, 15(2):266–277, 1967.

[57] F. Gorunescu, S.I. McClean, and P.H. Millard. A queueing model for bed-occupancy management and planning of hospitals. *The Journal of the Operational Research Society*, 53(1):19–24, 2002.

[58] Prachi Goyal, Pranabendu Misra, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Finding even subgraphs even faster. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, page 434, 2015.

[59] Linda Green. Queueing analysis in healthcare. In *Patient flow: reducing delay in healthcare delivery*, pages 281–307. Springer, 2006.

[60] Jacqueline Griffin, Shuangjun Xia, Siyang Peng, and Pinar Keskinocak. Improving patient flow in an obstetric unit. *Health Care Management Science*, 15(1):1–14, 2012.

[61] Murat M Günal and Michael Pidd. Discrete event simulation for performance modelling in health care: a review of the literature. *Journal of Simulation*, 4(1):42–51, 2010.

[62] Frank A Haight. Queueing with balking. *Biometrika*, 44(3/4):360–369, 1957.

[63] Marc Harper et al. python-ternary: Ternary plots in Python. 2015.

[64] Carl Hierholzer and Chr Wiener. Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, Mar 1873.

[65] J. Highsmith and A. Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[66] R. Holt. Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR)*, 4(3):179–196, 1972.

[67] N.P.C Hong, T. Crick, I.P. Gent, and L. Kotthoff. Top tips to make your research irreproducible. *arXiv preprint arXiv:1504.00062*, 2015.

[68] G. Hunt. Sequential arrays of waiting lines. *Operations Research*, 4(6):674–683, 1956.

[69] John D Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[70] Navid Izady. Appointment capacity planning in specialty clinics: A queueing approach. *Operations Research*, 63(4):916–930, 2015.

[71] J. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, 1957.

[72] James R Jackson. Jobshop-like queueing systems. *Management Science*, 10(1):131–142, 1963.

[73] RRP Jackson. Queueing systems with phase type service. *Journal of the Operational Research Society*, 5(4):109–120, 1954.

[74] Vikas Jha and Rajive Bagrodia. Simultaneous events and lookahead in simulation protocols. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 10(3):241–267, 2000.

[75] Rafael C. Jiménez, Mateusz Kuzak, Monther Alhamdoosh, Michelle Barker, Bérénice Batut, Mikael Borg, Salvador Capella-Gutierrez, Neil Chue Hong, Martin Cook, Manuel Corpas, Madison Flannery, Leyla Garcia, Josep Ll., Gelpí, Simon Gladman, Carole Goble, Montserrat González Ferreiro, Alejandra Gonzalez-Beltran, Philippa C. Griffin, Björn Grüning, Jonas Hagberg, Petr Holub, Rob Hooft, Jon Ison, Daniel S. Katz, Brane Leskošek, Federico López Gómez, Luis J. Oliveira, David Mellor, Rowland Mosbergen, Nicola Mulder, Yasset Perez-Riverol, Robert Pergl, Horst Pichler, Bernard Pope, Ferran Sanz, Maria V. Schneider, Victoria Stodden, Radosław Suchecki, Radka Svobodová Vařeková, Harry-Anton Talvik, Ilian Todorov, Andrew Treloar, Sonika Tyagi, Maarten van Gompel, Daniel Vaughan, Allegra Via, Xiaochuan Wang, and Nathan S. Watson-Haigh. Four simple recommendations to encourage best practices in research software. *F1000Research*, 6, 2017.

[76] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001–.

[77] P. Kawadkar, S. Prasad, and A.D. Dwivedi. Deadlock avoidance based on Banker's algorithm for waiting state processes. *International Journal of Innovative Science and Modern Engineering*, 2(12), 2014.

[78] Jau-Chuan Ke, Chia-Huang Wu, and Zhe George Zhang. Recent developments in vacation queueing models: a short survey. *International Journal of Operations Research*, 7(4):3–8, 2010.

[79] F. Kelly. Networks of queues with customers of different types. *Journal of Applied Probability*, 12(3):542–554, 1975.

[80] Maggie Kenney. Newport integrated older people pathway: Roll out business case. Technical report, PeopleToo, Sept 2015.

[81] Richard A Kilgore. Open source simulation modeling language SML. In *Proceedings of the 33nd Conference on Winter Simulation*, pages 607–613. IEEE Computer Society, 2001.

[82] S Kim and S. Kim. Differentiated waiting time management according to patient class in an emergency care centre using an open Jackson network integrated with pooling and prioritizing. *Annals of Operations Research*, pages 1–21, 2015.

[83] J. F. C. Kingman. The first Erlang century - and the next. *Queueing Systems*, 63(1):3–12, Nov 2009.

[84] P. Kirkpatrick and P.C. Bell. Simulation modelling: a comparison of visual interactive and traditional approaches. *European Journal of Operational Research*, 39(2):138–149, 1989.

[85] Vincent A. Knight, Geraint I. Palmer, and Nikoleta E. Glynatsi. Genetic algorithm exercise for an MSc hackathon using Ciw. `https://doi.org/10.5281/zenodo.836243`, jul 2017.

[86] Ernest Koenigsberg. Cyclic queues. *Journal of the Operational Research Society*, 9(1):22–35, 1958.

[87] N. Koizumi, E. Kuno, and T.E. Smith. Modeling patient flows using a queueing network with blocking. *Health Care Management Science*, 8(1):49–60, 2005.

[88] R. Korporaal, A. Ridder, P. Kloprogge, and R. Dekker. An analytic model for capacity planning of prisons in the Netherlands. *The Journal of the Operational Research Society*, 51(11):1228–1237, 2000.

[89] S. Kundu and I. Akyildiz. Deadlock buffer allocation in closed queueing networks. *Queueing Systems*, 4(1):47–56, 1989.

[90] C Lakshmi and Sivakumar Appa Iyer. Application of queueing theory in health care: A literature review. *Operations Research for Health Care*, 2(1-2):25–39, 2013.

[91] G. Latouche and M. Neuts. Efficient algorithmic solutions to exponential tandem queues with blocking. *SIAM Journal on Algebraic Discrete Methods*, 1(1):93–106, 1980.

[92] Averill M Law. *Simulation modeling and analysis.* McGraw-Hill, 2007.

[93] Hyo Kyung Lee, Albert J Musa, Philip A Bain, Kenneth Nelson, Christine Baker, and Jingshan Li. A queueing network model for analysis of patient transitions within hospitals. *IEEE Transactions on Automation Science and Engineering*, 2018.

[94] Austin J Lemoine. Networks of queues - a survey of equilibrium analysis. *Management Science*, 24(4):464–481, 1977.

[95] Lemur Heavy Industries. Coveralls. `https://coveralls.io`.

[96] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003.

[97] J. Leung and E. Lai. On minimum cost recovery form system deadlock. *IEEE Transactions on Computers*, 28(9):671–677, 1979.

[98] J. Leung and B. Monien. On the complexity of deadlock recovery. In *STACS 85*, pages 208–218. Springer, 1985.

[99] Hongwei Liao, Hao Zhou, and Stéphane Lafortune. Simulation analysis of multithreaded programs under deadlock-avoidance control. *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 703–715, 2011.

[100] J. Liebeherr and I. Akyildiz. Deadlock properties of queueing networks with finite capacities and multiple routing chains. *Queueing Systems*, 20(3-4):409–431, 1995.

[101] D.R. MacIver. Hypothesis. `https://hypothesis.readthedocs.org/`, 2017.

[102] A. Manzano-Santaella. From bed-blocking to delayed discharges: precursors and interpretations of a contested concept. *Health Services Management Research*, 23(3):121–127, 2010.

[103] O. Marchetti and A. Munier-Kordon. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research*, 197(2):532–540, 2009.

[104] Wes McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. SciPy Austin, TX, 2010.

[105] Daniel A Menascé and Richard R Muntz. Locking and deadlock detection in distributed data bases. *IEEE Transactions on Software Engineering*, (3):195–202, 1979.

[106] Mercurial-scm. Mercurial. `https://www.mercurial-scm.org/`.

[107] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, 2017.

[108] Toshimi Minoura. Deadlock avoidance revisited. *Journal of the ACM (JACM)*, 29(4):1023–1048, 1982.

[109] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

[110] Thomas Monks, Christine SM Currie, Bhakti Stephan Onggo, Stewart Robinson, Martin Kunc, and Simon JE Taylor. Strengthening the reporting of empirical simulation studies: Introducing the STRESS guidelines. *Journal of Simulation*, pages 1–13, 2018.

[111] James E Murphy. Resource allocation with interlock detection in a multi-task system. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part II*, pages 1169–1176. ACM, 1968.

[112] National Centre for Research Methods. Georgraphical referencing learning resources: Townsend deprivation index. `http://www.restore.ac.uk/geo-refer/36229dtuks00y19810000.php`.

[113] NICE. Standards and indicators. `https://www.nice.org.uk/standards-and-indicators?tab=qof`.

[114] Ronald Anthony Nosek Jr and James P Wilson. Queuing theory and customer satisfaction: a review of terminology, trends, and applications to pharmacy practice. *Hospital Pharmacy*, 36(3):275–279, 2001.

[115] Nuffield Foundation. Nuffield Research Placements. `http://www.nuffieldfoundation.org/nuffield-research-placements`, 2017.

[116] ONS. Death registrations summary tables - england and wales. `https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/deaths/datasets/deathregistrationssummarytablesenglandandwalesreferencetables/`.

[117] ONS. Lower layer super output areas (december 2011) full clipped boundaries in england and wales. `http://geoportal.statistics.gov.uk/datasets/da831f80764346889837c72508f046fa_0/`.

[118] ONS. Lower super output area mid-year population estimates (supporting information). `https://www.ons.gov.uk/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/lowersuperoutputareamidyearpopulationestimates/`.

[119] R. Onvural. Survey of closed queueing networks with blocking. *ACM Computing Surveys*, 22(2):83–121, 1990.

[120] R. Onvural and H.G. Perros. On equivalencies of blocking mechanisms in queueing networks with blocking. *Operations Research Letters*, 5(6):293–297, 1986.

[121] C. Osorio and M. Bierlaire. An analytic finite capacity queueing network model capturing the propagation of congestion and blocking. *European Journal of Operational Research*, 196(3):996–1007, 2009.

[122] Geraint I Palmer, Paul R Harper, and Vincent A Knight. Modelling deadlock in open restricted queueing networks. *European Journal of Operational Research*, 266(2):609–621, 2018.

[123] Geraint I. Palmer, Asyl L. Hawa, Vincent A. Knight, and Paul R. Harper. M/M/3 simulation models for the paper "Ciw: An open source discrete event simulation library.". `https://doi.org/10.5281/zenodo.848644`, August 2017.

[124] Geraint I Palmer, Vincent A Knight, Paul R Harper, and Asyl L Hawa. Ciw: An open-source discrete event simulation library. *Journal of Simulation*, pages 1–15, 2018.

[125] Geraint Ian Palmer. Code for thesis: Chapter 5 markov models of deadlock. Sep 2018.

[126] Geraint Ian Palmer. Code for thesis: Chapter 6 deadlock resolution. Sep 2018.

[127] Geraint Ian Palmer. Code for thesis: Chapter 7 modelling Stay Well Plans in gwent. Sep 2018.

[128] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[129] PeopleToo. OP pathway progress update august 2015. Technical report, Aug 2015.

[130] PeopleToo. Newport integrated older people's pathway: rapid evaluation - primary data analysis report. Technical report, May 2016.

[131] H.J.W. Percival. *Test-driven development with Python*. O'Reilly, 2014.

[132] H.G. Perros, A.A. Nilsson, and Y.C. Liu. Approximate analysis of product-form type queueing networks with blocking and deadlock. *Performance Evaluation*, 8(1):19–39, 1988.

[133] Patrick Peschlow and Peter Martini. A discrete-event simulation tool for the analysis of simultaneous events. In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, page 14. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[134] M Pidd. Object-orientation, discrete simulation and the three-phase approach. *The Journal of the Operational Research Society*, 46(3):362–374, 1995.

[135] Pocoo. Sphinx Python documentation generator. `http://www.sphinx-doc.org`.

[136] A. Prlić and J.B. Procter. Ten simple rules for the open development of scientific software. *PLoS Computational Biology*, 8(12), 2012.

[137] D. Procida. What nobody tells you about documentation. `https://www.divio.com/en/blog/documentation/`, 2017.

[138] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2005.

[139] Stanislaw Raczynski. *Modeling and simulation: the computer science of illusion.* John Wiley & Sons, 2014.

[140] Eric S Raymond. *The Cathedral & the Bazaar.* 1999.

[141] ReadtheDocs. Read the Docs. `https://readthedocs.org`.

[142] K. Rege. Multi-class queueing models for performance analysis of computer systems. *Sadhana*, 15(4-5):355–363, 1990.

[143] E. Reich. Waiting times when queues are in tandem. *The Annuls of Mathematical Statistics*, 28(3):768–773, 1957.

[144] S. Reveliotis. Coordinating autonomy. *IEEE Robotics & Automation Magazine*, 22(2):77–94, 2015.

[145] S. Reveliotis. Real-time management of complex resource allocation systems: necessity, achievements and further challenges. *IFAC-PapersOnLine*, 48(7):50–57, 2015.

[146] Stewart Robinson. Discrete-event simulation: from the pioneers to the present, what next? *Journal of the Operational Research Society*, 56(6):619–629, 2005.

[147] Stewart Robinson. *Simulation: the practice of model development and use.* Palgrave Macmillan, 2014.

[148] Nayan B Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[149] G.K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10):1–4, 2013.

[150] L. C Schmidt and J. Jackman. Modeling recirculating conveyors with blocking. *European Journal of Operational Research*, 124(2):422–436, 2000.

[151] Thomas J Schriber, Daniel T Brunner, and Jeffrey S Smith. Inside discrete-event simulation software: how it works and why it matters. In *Simulation Conference (WSC), 2013 Winter*, pages 424–438. IEEE, 2013.

[152] Daniel A Schult and P Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, volume 2008, pages 11–16, 2008.

[153] SIMUL8 Corporation. SIMUL8. `https://www.simul8.com/`, 2017.

[154] Mukesh Singhal. Deadlock detection in distributed systems. *Computer*, 22(11):37–48, 1989.

[155] Slashdot Media. SourceForge. `https://sourceforge.net/`.

[156] StatWales. Welsh Index of Multiple Deprivation 2014 by rank and lower super output area. `https://statswales.gov.wales/Catalogue/Community-Safety-and-Social-Inclusion/Welsh-Index-of-Multiple-Deprivation/WIMD-2014/wimd2014/`.

[157] W. Stewart. *Probability, markov chains, queues, and simulation.* Princeton University Press, 2009.

[158] R. Sutton and A. Barto. *Reinforecement learning: an introduction.* MIT Press, 1998.

[159] C. Szepesvári. *Algorithms for reinforcement learning.* Morgan & Claypool Publishers, 2010.

[160] Y. Takahashi, H. Miyahara, and T. Hasegawa. An approximation method for open restricted queueing networks. *Operations Research*, 28(3):594–602, 1980.

[161] S.T. Tan. *Calculus.* Brooks/Cole/Cengage Learning, 2009.

[162] Team SimPy. SimPy. `https://simpy.readthedocs.io/`, 2017.

[163] The AnyLogic Company. AnyLogic. `https://www.anylogic.com/`, 2017.

[164] The Online Encyclopedia of Integer Sequences. `https://oeis.org/`.

[165] The Operational Research Society. What is O.R.? Accessed: 2018-06-25.

[166] The PyPy developers. PyPy. `https://pypy.org/`, 2017.

[167] The Python Software Foundation. Python Package Index. `https://pypi.python.org/pypi`.

[168] The Python Software Foundation. Python Wiki. `https://wiki.python.org/moin/PythonTestingToolsTaxonomy`.

[169] The Python Software Foundation. Python 3.5.1. `www.python.org`, 2015.

[170] Travis CI. Travis CI. `https://travis-ci.org`.

[171] K Udagawa and G Nakamura. On a queue in which joining customers give up their services halfway. *J. Operations Res. Soc. Japan*, 1:186–197, 1957.

[172] Kanehisa Udagawa and Gisaku Nakamura. On a certain queuing system. In *PKodai Mathematical Seminar Reports*, volume 8, pages 117–124. Department of Mathematics, Tokyo Institute of Technology, 1956.

[173] P.T. Vanberkel, R.J. Boucherie, E.W. Hans, J.L. Hurink, and N. Litvak. A survey of health care models that encompass multiple departments. 2009.

[174] S. Venkatesh and J. Smith. A graph-theoretic, linear-time scheme to detect and resolve deadlocks in flexible manufacturing cells. *Journal of Manufacturing Systems*, 22(3):220–238, 2003.

[175] S. Venkatesh and J. Smith. An evaluation of deadlock-handling strategies in semiconductor cluster tools. *IEEE Transactions on Semiconductor Manufacturing*, 18(1):197–201, 2005.

[176] S. Venkatesh, J. Smith, B. Deuermeyer, and G. Curry. Deadlock detection and resolution for discrete-event simulation: multiple-unit seizes. *IIE Transactions*, 30(3):201–206, 1998.

[177] Julie Vile. *Time-dependent stochastic modelling for predicting demand and scheduling of emergency medical services.* PhD thesis, Cardiff University, 2013.

[178] I.F.A. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006.

[179] N. Viswanadham, Y. Narahari, and T.L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, 1990.

[180] Georg Von Krogh and Eric Von Hippel. The promise of research on open source software. *Management Science*, 52(7):975–983, 2006.

[181] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[182] Michael Waskom, Olga Botvinnik, Paul Hobson, John B. Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, Tom Augspurger, Tal Yarkoni, Tobias Megies, Luis Pedro Coelho, Daniel Wehner, cynddl, Erik Ziegler, diego0020, Yury V. Zaytsev, Travis Hoppe, Skipper Seabold, Phillip Cloud, Miikka Koskinen, Kyle Meyer, Adel Qalieh, and Dan Allan. seaborn: v0.5.0 (November 2014), nov 2014.

[183] Welsh Government. Local authority population projections. `http://gov.wales/statistics-and-research/local-authority-population-projections/?lang=en/`.

[184] Welsh Government. Welsh index of multiple deprivation (WIMD) 2014. `http://wimd.wales.gov.uk`.

[185] Frederick Wieland. The threshold of event simultaneity. In *ACM SIGSIM Simulation Digest*, volume 27, pages 56–59. IEEE Computer Society, 1997.

[186] G. Wilson, D.A. Aruliah, C. T. Brown, N.P.C. Hong, M. Davis, R.T. Guy, S.H.D. Haddock, K.D. Huff, I.M. Mitchell, M.D Plumbley, B. Waugh, E.P. White, and P. Wilson. Best practices for scientific computing. *PLoS Biology*, 12(1):e1001745, 2014.

[187] Robin J Wilson. *Introduction to graph theory.* Longman Scientific and Technical, 1985.

[188] D Worthington. Reflections on queue modelling from the last 50 years. *Journal of the Operational Research Society*, 60(1):S83–S92, 2009.

[189] M. Ziemann, Y. Eren, and A. El-Osta. Gene name errors are widespread in the scientific literature. *Genome Biology*, 17(1):177, 2016.

Appendix A

Simulation Models for Comparison

All models can also be found at [123].

C++ Model

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;

double runTrial(int seed, double arrivalRate, double serviceRate, int
↪  numberOfServers, double maxSimTime, double warmup){
  int i;
  double outcome, r1, r2, serviceTime, serviceStartDate, serviceEndDate,
   ↪  wait;
  double sumWaits = 0.0, arrivalDate = 0.0;
  vector<double> serversEnd;
  vector<double> temp;
  vector<double> waits;
  vector<vector<double> > records;
  srand(seed);

  for(i = 0; i < numberOfServers; ++i){
    serversEnd.push_back(0);
  }

  while(arrivalDate < maxSimTime){
    r1 = ((double)rand() / (double)RAND_MAX);
    r2 = ((double)rand() / (double)RAND_MAX);
    while (r1 == 0.0 || r2 == 0.0 || r1 == 1.0 || r2 == 1.0){
      r1 = ((double)rand() / (double)RAND_MAX);
      r2 = ((double)rand() / (double)RAND_MAX);
    }
    arrivalDate += (-log(r1))/arrivalRate;
    serviceTime = (-log(r2))/serviceRate;
    serviceStartDate = max(arrivalDate,
     ↪  (*min_element(serversEnd.begin(), serversEnd.end())));
    serviceEndDate = serviceStartDate + serviceTime;
    wait = serviceStartDate - arrivalDate;
    serversEnd.push_back(serviceEndDate);
    serversEnd.erase(min_element(serversEnd.begin(), serversEnd.end()));
    temp.push_back(arrivalDate);
    temp.push_back(wait);
    records.push_back(temp);
    temp.clear();
  }

  for(i = 0; i < records.size(); ++i){
    if(records[i][0] > warmup){
      waits.push_back(records[i][1]);
    }
  }

  for(i = 0; i < waits.size(); ++i){
    sumWaits += waits[i];
  }

  outcome = (sumWaits) / (waits.size());
  return outcome;
}

int main(int argc, char **argv){
  int i, seed;
  double solution;
  int numberOfServers = 3;
  int numberOfTrials = 20;
  double arrivalRate = 10.0;
  double serviceRate = 4.0;
  double maxSimTime = 800.0;
  double warmup = 100.0;
  double sumMeanWaits = 0.0;
  vector<double> meanWaits;

  for(seed = 0; seed < numberOfTrials; ++seed ){
    meanWaits.push_back(runTrial(seed, arrivalRate, serviceRate,
     ↪  numberOfServers, maxSimTime, warmup));
  }

  for(i = 0; i < meanWaits.size(); ++i){
    sumMeanWaits += meanWaits[i];
  }
  solution = (sumMeanWaits) / (meanWaits.size());
}
```

SimPy Model

```
import simpy
import random

arrival_rate = 10.0
number_of_servers = 3
service_rate = 4.0
max_simulation_time = 800
warmup = 100
num_trials = 20

def source(env, arrival_rate, service_rate, server, records):
    """
    Source generates customers randomly
    """
    while True:
        c = customer(env, server, service_rate, records)
        env.process(c)
        t = random.expovariate(arrival_rate)
        yield env.timeout(t)

def customer(env, server, service_rate, records):
    """
    Customer arrives, is served and leaves.
    """
    arrive = env.now
    with server.request() as req:
        results = yield req
        wait = env.now - arrive
        records.append((env.now, wait))
        tib = random.expovariate(service_rate)
        yield env.timeout(tib)

def run_trial(seed, arrival_rate, service_rate, number_of_servers,
              max_simulation_time, warmup):
    """
    Run one trial of the simulation, returning the average waiting time
    """
    random.seed(seed)
    records = []
    env = simpy.Environment()
    server = simpy.Resource(env, capacity=number_of_servers)
    env.process(source(env, arrival_rate, service_rate, server, records))
    env.run(until=max_simulation_time)
    waiting_times = [r[1] for r in records if r[0] > warmup]
    return sum(waiting_times) / len(waiting_times)

mean_waits = [run_trial(
  s, arrival_rate, service_rate, number_of_servers,
  max_simulation_time, warmup) for s in range(num_trials)]

average_waits = sum(mean_waits) / len(mean_waits)
```

Ciw Model

```
import ciw

max_simulation_time = 800
warmup = 100
num_trials = 20

N = ciw.create_network(
    Arrival_distributions=[['Exponential', 10.0]],
    Service_distributions=[['Exponential', 4.0]],
    Number_of_servers=[3]
)

def run_trial(s, max_simulation_time, warmup):
    """
    Run one trial of the simulation, returning the average waiting time
    """
    ciw.seed(s)
    Q = ciw.Simulation(N)
    Q.simulate_until_max_time(max_simulation_time)
    recs = Q.get_all_records()
    waits = [r.waiting_time for r in recs if r.arrival_date > warmup]
    return sum(waits) / len(waits)

mean_waits = [
    run_trial(s, max_simulation_time, warmup) for s in range(num_trials)
]

average_waits = sum(mean_waits) / len(mean_waits)
```

Appendix B

Illustrative Example I

The state space Z has 22 states:

$$z_{01} = \begin{array}{c}\begin{pmatrix} (1) & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \\ (\ 1 \quad 0\ )\end{array} \qquad z_{07} = \begin{array}{c}\begin{pmatrix} (3) & \emptyset \\ (1,2) & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{13} = \begin{array}{c}\begin{pmatrix} \emptyset & (2) \\ \emptyset & (1,3) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{19} = \begin{array}{c}\begin{pmatrix} \emptyset & (2) \\ (1) & (3) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

$$z_{02} = \begin{array}{c}\begin{pmatrix} (1) & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \\ (\ 1 \quad 1\ )\end{array} \qquad z_{08} = \begin{array}{c}\begin{pmatrix} (3) & \emptyset \\ (1) & (2) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{14} = \begin{array}{c}\begin{pmatrix} \emptyset & (1) \\ (2,3) & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{20} = \begin{array}{c}\begin{pmatrix} \emptyset & (2) \\ (3) & (1) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

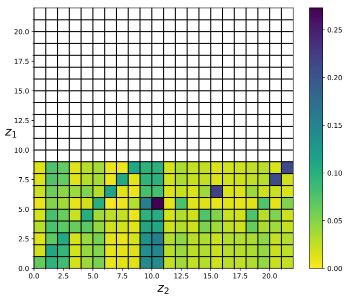
$$z_{03} = \begin{array}{c}\begin{pmatrix} (1) & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{09} = \begin{array}{c}\begin{pmatrix} (3) & \emptyset \\ (2) & (1) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{15} = \begin{array}{c}\begin{pmatrix} \emptyset & (2) \\ (1,3) & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{21} = \begin{array}{c}\begin{pmatrix} \emptyset & (3) \\ (1) & (2) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

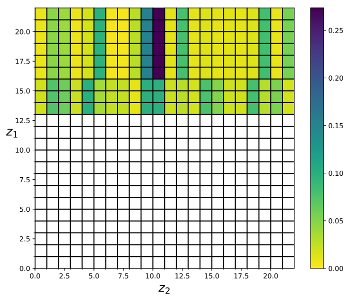
$$z_{04} = \begin{array}{c}\begin{pmatrix} (2) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\ 1 \quad 1\ )\end{array} \qquad z_{10} = \begin{array}{c}\begin{pmatrix} \emptyset & \emptyset \\ \emptyset & (1,2) \end{pmatrix} \\ (\ 0 \quad 2\ )\end{array} \qquad z_{16} = \begin{array}{c}\begin{pmatrix} \emptyset & (3) \\ (1,2) & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{22} = \begin{array}{c}\begin{pmatrix} \emptyset & (3) \\ (2) & (1) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

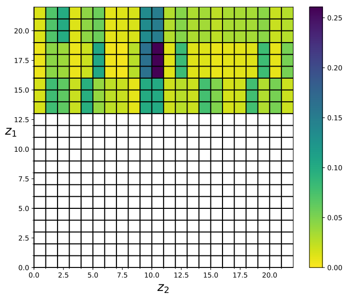
$$z_{05} = \begin{array}{c}\begin{pmatrix} (2) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{11} = \begin{array}{c}\begin{pmatrix} \emptyset & \emptyset \\ \emptyset & (1,2) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{17} = \begin{array}{c}\begin{pmatrix} \emptyset & (1) \\ (2) & (3) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

$$z_{06} = \begin{array}{c}\begin{pmatrix} (2) & \emptyset \\ \emptyset & (1) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{12} = \begin{array}{c}\begin{pmatrix} \emptyset & (1) \\ \emptyset & (2,3) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array} \qquad z_{18} = \begin{array}{c}\begin{pmatrix} \emptyset & (1) \\ (3) & (2) \end{pmatrix} \\ (\ 1 \quad 2\ )\end{array}$$

Figures B.1 and B.2 give the simulated transition probabilities and rewards for each action (over 10,000 runs). Blank spaces indicate infeasible action-state pairs.


(a) $a = (1, 1)$


(b) $a = (1, 2)$


(c) $a = (2, 1)$

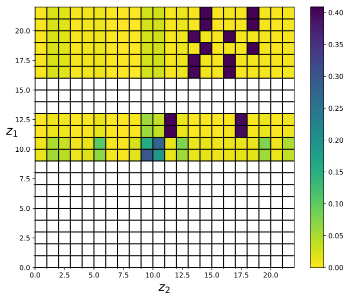
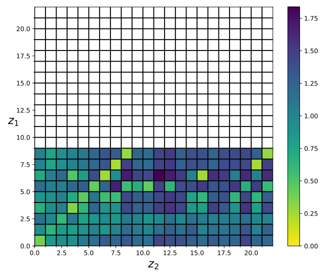
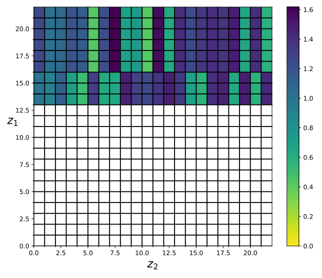
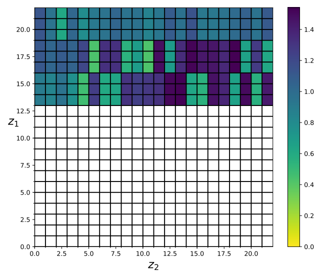

(d) $a = (2, 2)$

Figure B.1: Simulated values of $p(z_1, z_2, a)$ for illustrative example I.


(a) $a = (1, 1)$


(b) $a = (1, 2)$


(c) $a = (2, 1)$

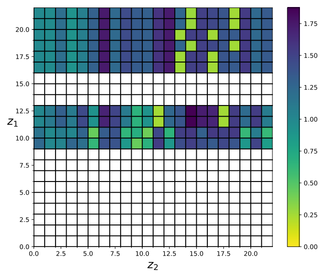

(d) $a = (2, 2)$

Figure B.2: Simulated values of $\omega(z_1, z_2, a)$ for illustrative example I.

Appendix C

Illustrative Example II

The state space has 43 states, listed below:

$$s_{01} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 0 \quad 4\ )\end{array} \qquad s_{08} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 1 \quad 4\ )\end{array} \qquad s_{15} = \begin{array}{c}\begin{pmatrix}\emptyset & (1,2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 3 \quad 4\ )\end{array} \qquad s_{22} = \begin{array}{c}\begin{pmatrix}(1,2) & \emptyset\\ \emptyset & \emptyset\end{pmatrix}\\ (\ 6 \quad 0\ )\end{array}$$

$$s_{02} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 1 \quad 4\ )\end{array} \qquad s_{09} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 2 \quad 4\ )\end{array} \qquad s_{16} = \begin{array}{c}\begin{pmatrix}\emptyset & (1,2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 4 \quad 4\ )\end{array} \qquad s_{23} = \begin{array}{c}\begin{pmatrix}(1,2) & \emptyset\\ \emptyset & \emptyset\end{pmatrix}\\ (\ 6 \quad 1\ )\end{array}$$

$$s_{03} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 2 \quad 4\ )\end{array} \qquad s_{10} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 3 \quad 4\ )\end{array} \qquad s_{17} = \begin{array}{c}\begin{pmatrix}\emptyset & (1,2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 5 \quad 4\ )\end{array} \qquad s_{24} = \begin{array}{c}\begin{pmatrix}(1,2) & \emptyset\\ \emptyset & \emptyset\end{pmatrix}\\ (\ 6 \quad 2\ )\end{array}$$

$$s_{04} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 3 \quad 4\ )\end{array} \qquad s_{11} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 4 \quad 4\ )\end{array} \qquad s_{18} = \begin{array}{c}\begin{pmatrix}\emptyset & (1,2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{25} = \begin{array}{c}\begin{pmatrix}(1,2) & \emptyset\\ \emptyset & \emptyset\end{pmatrix}\\ (\ 6 \quad 3\ )\end{array}$$

$$s_{05} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 4 \quad 4\ )\end{array} \qquad s_{12} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 5 \quad 4\ )\end{array} \qquad s_{19} = \begin{array}{c}\begin{pmatrix}(1) & \emptyset\\ \emptyset & (2)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{26} = \begin{array}{c}\begin{pmatrix}(1,2) & \emptyset\\ \emptyset & \emptyset\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array}$$

$$s_{06} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 5 \quad 4\ )\end{array} \qquad s_{13} = \begin{array}{c}\begin{pmatrix}\emptyset & (1)\\ \emptyset & (2)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{20} = \begin{array}{c}\begin{pmatrix}(1) & (2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{27} = \begin{array}{c}\begin{pmatrix}(1,3) & \emptyset\\ (2) & \emptyset\end{pmatrix}\\ (\ 6 \quad 1\ )\end{array}$$

$$s_{07} = \begin{array}{c}\begin{pmatrix}\emptyset & \emptyset\\ \emptyset & (1)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{14} = \begin{array}{c}\begin{pmatrix}\emptyset & (1,2)\\ \emptyset & (3)\end{pmatrix}\\ (\ 2 \quad 4\ )\end{array} \qquad s_{21} = \begin{array}{c}\begin{pmatrix}(2) & (1)\\ \emptyset & (3)\end{pmatrix}\\ (\ 6 \quad 4\ )\end{array} \qquad s_{28} = \begin{array}{c}\begin{pmatrix}(1,3) & \emptyset\\ (2) & \emptyset\end{pmatrix}\\ (\ 6 \quad 2\ )\end{array}$$

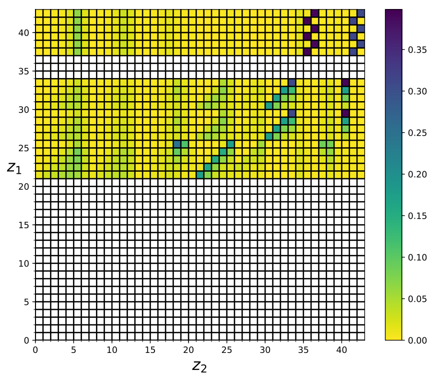
$$s_{29} = \begin{array}{c} \begin{pmatrix} (1,3) & \emptyset \\ (2) & \emptyset \end{pmatrix} \\ (\,6 \quad 3\,) \end{array} \qquad s_{33} = \begin{array}{c} \begin{pmatrix} (2,3) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 3\,) \end{array} \qquad s_{37} = \begin{array}{c} \begin{pmatrix} \emptyset & (2,3) \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{41} = \begin{array}{c} \begin{pmatrix} (2) & (3) \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array}$$

$$s_{30} = \begin{array}{c} \begin{pmatrix} (1,3) & \emptyset \\ (2) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{34} = \begin{array}{c} \begin{pmatrix} (2,3) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{38} = \begin{array}{c} \begin{pmatrix} (1) & (2) \\ (3) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{42} = \begin{array}{c} \begin{pmatrix} (3) & (1) \\ (2) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array}$$

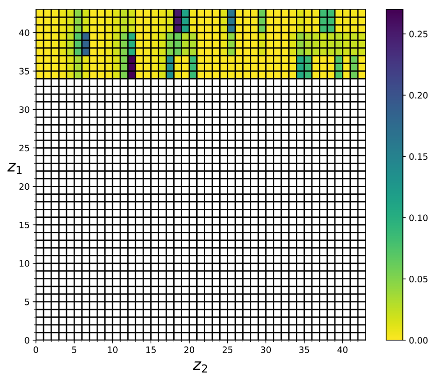
$$s_{31} = \begin{array}{c} \begin{pmatrix} (2,3) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 1\,) \end{array} \qquad s_{35} = \begin{array}{c} \begin{pmatrix} \emptyset & (1,2) \\ (3) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{39} = \begin{array}{c} \begin{pmatrix} (1) & (3) \\ (2) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{43} = \begin{array}{c} \begin{pmatrix} (3) & (2) \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array}$$

$$s_{32} = \begin{array}{c} \begin{pmatrix} (2,3) & \emptyset \\ (1) & \emptyset \end{pmatrix} \\ (\,6 \quad 2\,) \end{array} \qquad s_{36} = \begin{array}{c} \begin{pmatrix} \emptyset & (1,3) \\ (2) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array} \qquad s_{40} = \begin{array}{c} \begin{pmatrix} (2) & (1) \\ (3) & \emptyset \end{pmatrix} \\ (\,6 \quad 4\,) \end{array}$$

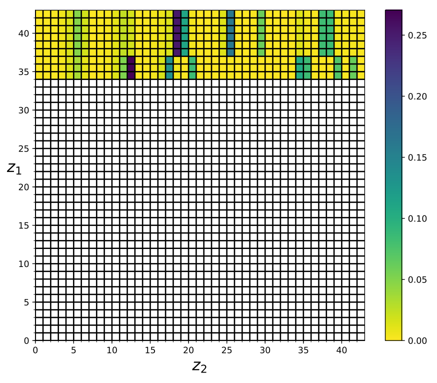
Figures C.1 and C.2 give the simulated transition probabilities and rewards for each action (over 10,000 runs). Blank spaces indicate infeasible action-state pairs.



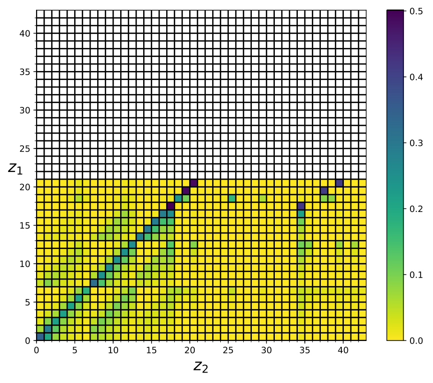
(a) $a = (1,1)$.



(b) $a = (1,2)$.

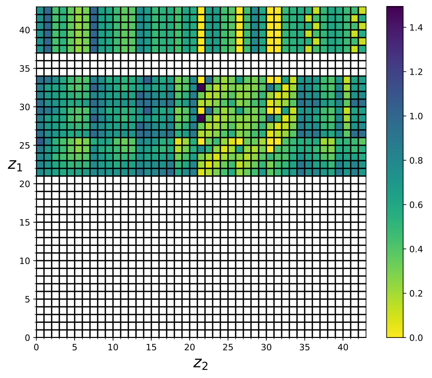


(c) $a = (2,1)$.

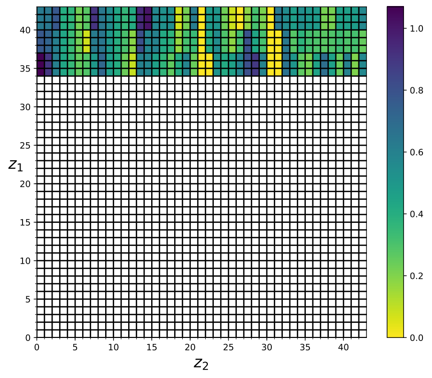


(d) $a = (2,2)$.

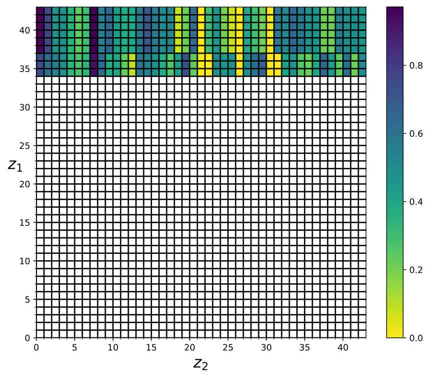
Figure C.1: Simulated values of $p(z_1, z_2, a)$ for illustrative example II.



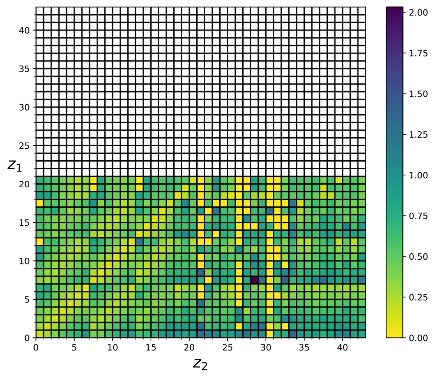
(a) $a = (1, 1)$



(b) $a = (1, 2)$



(c) $a = (2, 1)$



(d) $a = (2, 2)$

Figure C.2: Simulated values of $\omega(z_1, z_2, a)$ for illustrative example II.